

The Asymmetry Engine, Chapter 2: Checking is cheap

Martin Roark

Chapter 2 of *The Asymmetry Engine: How AI Closes the Gaps, and What Stays Scarce*, October 2026. The online edition, with the interactive figures and the links between chapters, is at <https://future-seems-so-good.com/blog/the-asymmetry-engine#verification>. License: CC BY 4.0.

2:1 Where checking a solution is far cheaper than finding one, and solved problems can be built backward from their answers, the training signal for a machine costs almost nothing, so those task families fall to machines first. Hard solved problems exist only if one-way functions do, the useful ones sit near a phase boundary, and once generation is nearly free, the check becomes the scarce input.

2:2 The ratio is already large in a newspaper puzzle. Checking a filled 9×9 Sudoku grid against its rules takes 243 cell reads; finding the grid from 26 clues took a naive search a median of 593,190 reads, about 2,400 checks (Experiment 2.1). Verification is the first of the four speeds named in §0.3: in the master equation it is the term v_k , which multiplies the closing rate of each gap, and a cheap, sound check is what raises it.

2.1 The coefficient

2:3 The verification asymmetry measures how much cheaper checking is than finding, and it belongs to a task and a solver together, so training turns it down.

2:4

Definition 2.1 (Verification asymmetry). For a task family with distribution $\mathcal{D}$ and a given solver, the **verification asymmetry** of a task x is

$$\alpha(x) = \frac{c_{\text{sol}}(x)}{c_{\text{ver}}(x)},$$

the cost of finding an accepted solution from x alone over the cost of checking a candidate. The family is asymmetric when $\mathbb{E}_{\mathcal{D}}[\alpha] \gg 1$. The ratio is relative to the solver: largest for a weak policy, and falling as the policy learns.

2:5 This is the NP picture. For a problem in NP a proposed solution can be checked in polynomial time, while finding one may take exponential time unless $P = NP$. A reward function is a verifier, so reinforcement learning needs only this ratio: it pays c_{ver} for each signal and collects what finding would have cost. Supervised learning needs the solution itself, a second product priced in (2.1).

2:6

"It's really hard to generate a correct solution, but it's much easier to recognize when you have one. I think all problems exist on the spectrum from really easy to verify relative to generation, like a Sudoku puzzle, versus just as hard to verify as it is to generate a solution, like naming the capital of Bhutan." Noam Brown of OpenAI, on Sequoia's [Training Data](#) podcast.

2:7 The position of a task on that spectrum moves with the solver. Against a naive backtracking search, a Sudoku at its hardest has an α of about 2,400, and against a search that branches on the most constrained cell, 113 to 164 (Experiment 2.1); against a solver that propagates constraints it is 10 to 20 (Figure 2.3). Every gain in the policy lowers c_{sol} and leaves c_{ver} where it was, so a family is most asymmetric for the weakest learner and least for one that has mastered it.

2:8 A large α helps only a learner that can use it. Song and colleagues found that "despite the exponential computational complexity separation between generation and verification in generalized sudoku, most models fail to self-improve" ([Mind the Gap](#)). The ratio pays when the verifier is supplied from outside and the learner already reaches an accepted answer some of the time, the two conditions of Proposition 2.2.

2.2 Sudoku, measured in one unit

2:9 Counted in one unit, the cell read, the three costs of a 9×9 Sudoku come out $c_{\text{ver}} < c_{\text{con}} \ll c_{\text{sol}}$. A check reads the nine cells of each of the 27 rows, columns and boxes, 243 reads whatever the puzzle; building a solved puzzle backward costs about a dozen checks, because each placement scans its row, column and box; and search can cost thousands of checks (Experiment 2.1).

2:10

Experiment 2.1 (Sudoku, measured).

Setup: 9×9 Sudoku with every cost counted in cell reads; ten random puzzles at each of ten levels from 10 to 64 blanks, medians reported; method and tables in §B.1.

Parameters: construction fills a grid by randomized backtracking and erases cells at random; a naive row-major backtracker, and a minimum-remaining-values (MRV) solver that branches on the cell with the fewest candidates; a uniqueness proof searches for a second solution.

Result: construction costs 2,795–3,497 reads at every level, 11.5–14.4 checks. Naive search rises from 5,805 reads at 30 blanks to 593,190 at 55 ($\alpha = 2,441$) and falls to 278,924 at 64 ($\alpha = 1,148$). MRV cuts search 3 to 22 times and rises without a peak, from $\alpha = 49$ at 45 blanks to 113 at 55 and 164 at 64. A proper puzzle, with uniqueness proved after each erasure, costs 19,562 reads to build at 30 blanks and 468,437 at 55, as much as naive solving.

Shows: construction is cheap and still costs more than a check; α is a dial set by the solver; uniqueness is the hidden cost, and a verifier that checks the rules avoids it.

2:11 Hardness peaks near a phase boundary, and where it peaks depends on the solver. Half the randomly carved puzzles of Experiment 2.1 have a unique solution at 30 blanks, a fifth at 45 and none from 50 on. Naive search peaks at 55, just past that edge, where solutions are few but more than one; further out any of many solutions is accepted and search gets cheaper. The MRV solver has no peak, so the peak belongs to the solver as much as to the

puzzles. Cheeseman, Kanefsky and Taylor found the shape across NP-complete problems: "the hard problems occur at a critical value" of an order parameter, where "the probability of a solution changes abruptly from near 0 to near 1" (IJCAI 1991).

2:12 Uniqueness is the hidden cost. A newspaper needs a proper puzzle, one with exactly one solution, and deciding whether a puzzle has a second solution is ASP-complete, hence NP-complete for generalized $n^2 \times n^2$ Sudoku (Yato and Seta 2003). In Experiment 2.1 a uniqueness proof at 55 blanks costs about 115 checks, and building a proper puzzle costs 0.8 to 3.4 naive solves. The familiar line that puzzles are easy to make and hard to solve holds for solved grids and fails for proper ones: quality control puts a solver back inside the generator.

2:13 Training needs no uniqueness. A reward that accepts any completion satisfying the constraints is correct by the rules, so a backward-built puzzle with several solutions is still a valid task. The design rule is to *verify against the rules, not against the key*, and it carries over: run the tests rather than comparing with a reference solution, and check a proof in the kernel rather than comparing strings. A verifier that compares with a stored answer is only as sound as the answer is hidden (2:59).

2:14 Generators built this way are standard practice. Reasoning Gym ships "over 100 data generators and verifiers" with adjustable difficulty (Stojanovski et al. 2025); each of SynLogic's 35 tasks, Sudoku among them, pairs generation code with a rule-based verifier (SynLogic); and Logic-RL generalized from 5,000 generated logic puzzles to the AIME and AMC mathematics competitions (Xie et al. 2025).

2:15

Interactive in the online edition: <https://future-seems-so-good.com/blog/the-asymmetry-engine#fig:sudoku>

Figure 2.1 (interactive). Checking against solving on one grid, counted in cell reads. Press Check and the counter stops at 243 reads, whatever the puzzle; press Solve and naive search places and erases digits on a 55-blank puzzle until its counter passes half a million. Then drag the blanks slider from 10 to 64 and watch the check stay flat while naive search climbs to its peak at 55 blanks and falls past the edge of uniqueness, and the MRV curve keeps rising.

2.3 Building problems backward

2:16 Backward construction turns a cheap forward map into an unlimited supply of solved problems: start from the answer and compute a question it answers. Its leverage is the cost of search over the cost of building and checking, and its price is the distance between the problems it builds and the problems the world poses.

2:17

Definition 2.2 (Backward constructor). **Backward construction** builds a solved pair from its solution: draw y from a prior the designer chooses, then compute a problem x with $y \in \mathcal{Y}^*(x)$ by construction, at cost c_{con} . A backward constructor is a cheap sampler of such pairs. The designer chooses the prior over solutions; the world chooses the test distribution.

2:18 A pair found by forward search costs c_{sol} ; built backward and checked once, it costs $c_{\text{con}} + c_{\text{ver}}$; a reward for reinforcement learning costs c_{ver} . Against forward search the two products have leverage

2:19

$$\Lambda_{\text{sup}} = \frac{c_{\text{sol}}}{c_{\text{con}} + c_{\text{ver}}}, \quad \Lambda_{\text{RL}} = \alpha = \frac{c_{\text{sol}}}{c_{\text{ver}}} \quad (2.1)$$

2:20 On the Sudoku of Experiment 2.1 the first is about a thirteenth of the second, because construction costs about a dozen checks; at the naive searcher's peak it is still about 180.

2:21 The price is distributional. Constructed problems follow whatever law the forward map produces, $\mathcal{D}_{\text{con}}$, while the world poses problems from $\mathcal{D}$, and a policy's accuracy falls in the move from one to the other by at most the total variation distance between the two laws:

2:22

$$\text{Acc}_{\mathcal{D}}(f) \geq \text{Acc}_{\mathcal{D}_{\text{con}}}(f) - d_{\text{TV}}(\mathcal{D}_{\text{con}}, \mathcal{D}) \quad (2.2)$$

▼ *Derivation: The total-variation bound*

The difference between the expectations of a loss with values in $[0, 1]$ under the two laws is the integral of the loss against the difference of the laws, which lies between $-d_{\text{TV}}$ and d_{TV} ; the indicator of a correct answer gives (2.2). A policy that is right exactly where $\mathcal{D}_{\text{con}}$ puts more mass than $\mathcal{D}$ attains the bound, so without assumptions on the loss it cannot be improved.

2:23 Integration shows the price cleanly. Lample and Charton built training pairs by differentiating random functions, since "differentiation is always possible and extremely fast," and their backward-trained model scored 27.5% on integrals generated forward, while a forward-trained model scored 17.2% on the backward set ([Lample and Charton 2019](#)). The backward-trained model had learned "that integration tends to shorten expressions, a property that does not hold for FWD samples." Adding integration-by-parts pairs raised forward accuracy to 56.1%, and adding forward pairs to 94.3%. One backward constructor is a biased sampler of the world's problems; a curriculum is a blend of constructors chosen to cover them.

2:24 The recipe is one idea with many cheap forward maps. The last row is the warning: a perfect one-way function is cheap to run backward and teaches nothing, because its inverse has no regularity to learn (2:39).

2:25

Domain	Forward map, solution to problem	The model learns	Source
Symbolic integration	differentiate a random function	to integrate	Lample and Charton 2019
Euclidean geometry	sample premises, deduce, trace the proof back	the proof	Trinh et al. 2024
Code reasoning	run a program on an input	the input, or the program	Zhao et al. 2025
Software repair	revert a pull request, or inject a bug	the fix	Yang et al. 2025
Instruction following	take a human text as the answer and write its prompt	to follow the prompt	Li et al. 2023
Chains of thought	give the answer and ask for the reasoning	to reason to the answer	Zelikman et al. 2022
Cryptography, the limit	hash a random input	to invert the hash	Impagliazzo 1995

2:26 Every row pays the distributional price in its own currency. AlphaGeometry's 100 million synthetic theorems, built by deduction and traceback, skew toward shorter proofs and are "not constrained by human aesthetic biases such as being symmetrical" (Trinh et al. 2024). SWE-smith built 50,000 repair tasks from 128 repositories for about \$1,360, and its bugs made by reverting real pull requests trained best (Yang et al. 2025); bugs injected on purpose are out of distribution, because they do not reflect "realistic development processes" (BugPilot).

2:27 A 9×9 grid has one size and understates the asymmetry, which for generalized Sudoku outgrows every polynomial unless P = NP. Boolean satisfiability lets the size grow. Planted 3-SAT is backward construction in its plainest form: draw a random assignment of true and false to n variables, keep only clauses the assignment satisfies, and the formula arrives with its solution.

2:28

Experiment 2.2 (Planted 3-SAT).

Setup: random 3-SAT formulas, plainly planted ones (only clauses a hidden assignment satisfies) and balanced ones, whose hiding rule removes the literals' bias toward the assignment; DPLL with the MOMs branching rule and with naive branching; costs in one unit, the operation; method and tables in §B.2.

Parameters: 23 clause densities from 2 to 7 at $n = 150$ variables, and n from 20 to 300 at density 4.26, with 25 formulas per point.

Result: construction and checking are straight lines in n , $24.1n$ and $6.70n$ operations ($43.2n$ and $7.20n$ balanced); balanced search cost grows $\times 1.046$ per variable and balanced $\alpha \times 1.036$, from 26 at $n = 50$ to 3,246 at $n = 190$, though a power law fits nearly as well; random formulas are hardest at density 4.25, beside the 4.258 at which half are satisfiable; plain planting leaks its solution (each literal agrees with it $4/7$ of the time) and is solved with 2.2% of a random formula's effort.

Shows: the asymmetry widens with size, hard problems sit at a phase boundary, and a constructor must be one-way for the solver that trains on it.

2:29 Construction and checking stay linear, and search does not. In Experiment 2.2, construction costs 3.6 checks with plain and 6.0 with balanced planting at every size, so building never costs a search, while search on balanced formulas multiplies by about 1.046 per added variable and α by about 1.036: α is 26 at 50 variables, 1,200 at 150 and 3,246 at 190. The exponential reading of that growth rests on known lower bounds for resolution proofs more than on these sizes, and the absolute value depends on what is counted: charging only propagation and writes, the balanced α at 190 variables is 409.

2:30 Hard problems sit at a phase boundary. The share of random formulas that are satisfiable in Experiment 2.2 crosses one half at density 4.258, against the asymptotic 4.267 that Mézard and Zecchina computed, and solving cost peaks there at 51 times its value for under-constrained formulas. One knob sets the difficulty of the whole generator, blanks in Sudoku and clause density here, and the costly problems occupy a narrow band of it.

2:31 Plain planting leaks its solution. Each kept clause is one of the seven sign patterns the hidden assignment satisfies, so each literal agrees with the assignment with probability $4/7$, and a branching rule that counts literals reads the bias. Balanced planting keeps a clause less often the more of its literals are true, discounting by $(\sqrt{5} - 1)/2$, the inverse golden ratio, for each extra true literal; at that discount a literal agrees with the assignment

exactly half the time (Jia, Moore and Strain 2007). Removing the bias restores the hardness, with search cost growing $\times 1.046$ per variable against $\times 1.017$, for 43 construction operations per variable instead of 24 (Experiment 2.2). Dense planted formulas, balanced ones included, still give way to spectral methods (Flaxman 2003), so "hard" means hard for a class of solvers. A constructor must be one-way for the solver that will train on it, because the designer chooses the prior over solutions and the solver can read its fingerprints.

2:32

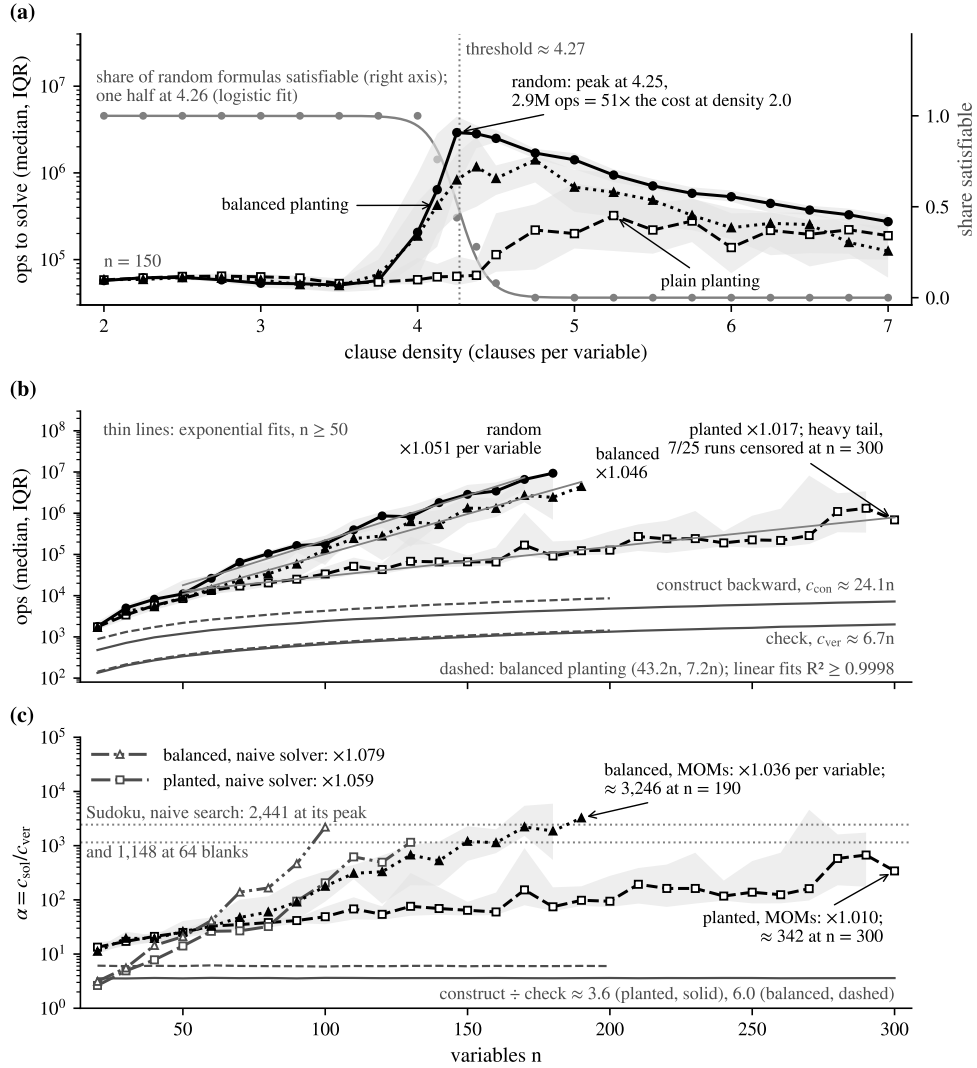


Figure 2.2. Panel (a) plots the median cost of solving random, plainly planted and balanced 3-SAT formulas of 150 variables against clause density, the number of clauses per variable, with the satisfiable share of random formulas; panel (b) plots the costs of construction c_{con} , checking c_{ver} and search c_{sol} against the number of variables at density 4.26; panel (c) plots $\alpha = c_{sol}/c_{ver}$ against the number of variables for the MOMs solver and for naive branching, with the naive Sudoku values as reference lines. Random formulas are hardest where the satisfiable share falls through one half; construction and checking rise as straight lines while balanced search climbs $\times 1.046$ per variable and its $\alpha \times 1.036$; with the MOMs solver, balanced planting passes the two Sudoku lines at 150 and 190 variables, and plain planting, which leaks its solution, stays far below them.

2:33 Software is the richest backward domain, because working code is a vast stock of solutions and breaking it is cheap. Mutation testing has seeded faults to grade test suites since 1978 (DeMillo, Lipton and Sayward); the LAVA corpus injected "thousands of bugs into eight real-world programs," each with an input that triggers it

(Dolan-Gavitt et al. 2016); Meta's self-play SWE-RL trains one model to inject bugs and repair them, gaining 10.4 points on SWE-bench Verified (Wei et al. 2025). A failing test or a triggering input is a short certificate, cheap and sound to check. The limit is again distributional: one vulnerability detector scored 68.26% F1 on an older benchmark and 3.09% on the realistic PrimeVul (Ding et al. 2024).

2:34 DARPA's AI Cyber Challenge makes the point in public. Its organizers injected synthetic vulnerabilities into 54 million lines of real open-source software and scored systems on finding and patching them (DARPA), and 12:26 gives the final's result. The benchmark can exist because the witness is checkable. Proving that no vulnerability is left has no short certificate, and §12.1 takes up what that does to the defender, while §12.4 counts what happens when finding outruns fixing.

2:35 Competitive programming has the same anatomy and the same catch. The recipe is to write a reference solution and a brute-force solver first, then derive the statement and tests: AutoCode's loop reaches 91.1% consistency with official judgments, and grandmaster-level competitors rated some of its problems "of contest quality" (AutoCode); rStar-Coder built 418,000 problems (rStar-Coder). The catch is the tests. 60% of the programs that pass the APPS benchmark's tests are wrong, and "TACO tests hurt the model's overall performance, while HardTests improves" it (He et al. 2025): a weak verifier makes reinforcement learning worse, because it rewards the wrong map. OpenAI's o3 found the remedy at test time, writing "simple brute-force solutions" to cross-check its optimized code (OpenAI 2025).

2.4 The one-way floor

2:36 Backward construction works because our world has a particular kind of hardness. Impagliazzo's "five worlds" sort possible universes by which hardness assumptions hold. In the one he calls Pessiland, "it is easy to generate many hard instances of NP problems. However, there is no way of generating hard solved instances of problems," because the map from a generator's random bits to its problem could be inverted to recover the solution; in Minicrypt, where one-way functions exist, a generator can pick an input, publish its image and keep the input as the answer (Impagliazzo 1995).

2:37

Proposition 2.1 (The one-way floor). A sampler of hard solved instances exists if and only if a one-way function exists. In Pessiland, where hard problems exist and one-way functions do not, the backward flywheel is impossible. A true one-way function cannot be learned, so useful constructed data sits in a band: hard for today's policy class, learnable by the next. Bug injection builds in a semantic property that Rice's theorem makes undecidable to detect in general, so its worst-case asymmetry is unbounded and the distribution of the injected bugs decides what it teaches.

▼ Proof

($\Leftarrow$) Given a one-way function, draw random bits and let the problem x be the function's value on them and the solution y the bits. The pair is solved by construction, no probabilistic polynomial-time solver finds a preimage of x with non-negligible probability, and construction costs one evaluation of the function, the order of a check.

($\Rightarrow$) Given a polynomial-time sampler that turns random bits into valid pairs (x, y) whose problems are hard on average, the map from the bits to x alone is one-way: an inverter that returned any bits with the same x

would let the sampler emit a valid solution of that problem, contradicting hardness. Success transfers exactly, because the sampler emits only valid pairs (the bits must be its only randomness). Weakly hard problems give a weak one-way function, which Yao's amplification makes strong.

The band follows from the definition: no polynomial-time learner inverts a one-way function, so its pairs carry nothing a learner can use. Rice's theorem makes every nontrivial semantic property of programs undecidable, while building a program with a known reachable bug is trivial. ■

2:38 So the cheap supply of hard training problems with known answers and public-key cryptography rest on one fact: without one-way functions there would be neither a secure internet nor such a supply. The root of training data and the root of cybersecurity are one root, which §12.1 follows to the defender's side.

2:39 One-wayness is necessary and not sufficient. A cryptographic hash is perfectly cheap to run backward and perfectly useless as training data, because its inverse has no regularity to learn. Useful constructed data comes from maps that are hard for the current learner and regular enough for the next, so its difficulty must track the model. Environment builders interviewed by Epoch target "a minimum pass rate of around 2-3%, or at least one success out of 64 or 128" rollouts ([Epoch AI, January 2026](#)), and a controlled study finds true gains from reinforcement learning only when the data are "calibrated to the model's edge of competence" ([Zhang, Neubig and Yue 2025](#)). Too easy gives no signal, too hard gives no gradient, and the band moves as the model learns.

2:40 The same logic reaches past decidability. John Wentworth's counterexample to the slogan that checking is always easier reads: "Generation problem: generate a program which halts. Verification problem: given a program, verify that it halts. The generation problem is trivial. The verification problem is uncomputable" ([LessWrong, 2022](#)). Bug injection turns that example into a data engine: building a program with a known bug is easy even where deciding whether an arbitrary program has one is impossible, so what the engine teaches depends on how its bugs are distributed.

2.5 Harvestable families

2:41 Three conditions decide which families machines take first.

2:42

Proposition 2.2 (Harvestable families). A task family is **harvestable**, able to sustain a flywheel of verifiable synthetic data, if and only if it has (i) a verifier that is cheap ($\mathbb{E}[\alpha] \gg 1$) and sound, (ii) a construction whose difficulty can be tuned to a phase boundary where α is large and the learner's pass rate stays positive, and (iii) a small distance d_{TV} between the constructed and the natural distribution. Families fall roughly in decreasing order of α and increasing order of shift: games and formal mathematics, code with tests, science checked by a digital twin, science checked by a wet lab, and last taste and values, where checking a candidate costs as much as producing it ($\alpha \approx 1$).

▼ Proof

A sketch. (i) If $c_{\text{ver}} \gtrsim c_{\text{sol}}$ then $\alpha \lesssim 1$, and no signal is cheaper than the search it would replace. If the verifier is unsound and its false accepts are reachable, a budget-bound optimizer raises verified success by reaching them (Proposition 1.1), and the flywheel manufactures reward hacking.

(ii) A constructor whose problems reveal their solutions emits easy problems (Proposition 2.1, Experiment 2.2), so without control of difficulty α collapses toward its value at the learner's competence. Past the learner's reach the base pass rate p_0 goes to zero, a first success takes $1/p_0$ draws on average, and the signal vanishes (Proposition 2.3). Only problems near the phase boundary are both hard and reachable.

(iii) A policy trained on the constructed law can lose up to d_{TV} of its accuracy on the world's, and the bound is attained ((2.2)).

Conversely, given (i)–(iii), each unit of construction budget yields usable pairs at a positive rate. Training raises the pass rate and lowers α for the learner, the curriculum re-aims at the boundary, and the loop runs until coverage saturates. ■

2:43 Figure 2.3 reads the proposition off the record. Its top rows, one-way functions, formal mathematics and games with rules, have cheap sound verifiers and a knob for difficulty. Code with tests has cheap verifiers whose soundness is in doubt. Below that the verifier is an expert, a panel or the world months later; where it is a wet lab, c_{ver} is measured in weeks and the verification asymmetry turns into a latency asymmetry (Definition 5.1, §14.3). Each step down gives up α or soundness, and the order of the rows is the order in which machines have taken the families.

2:44

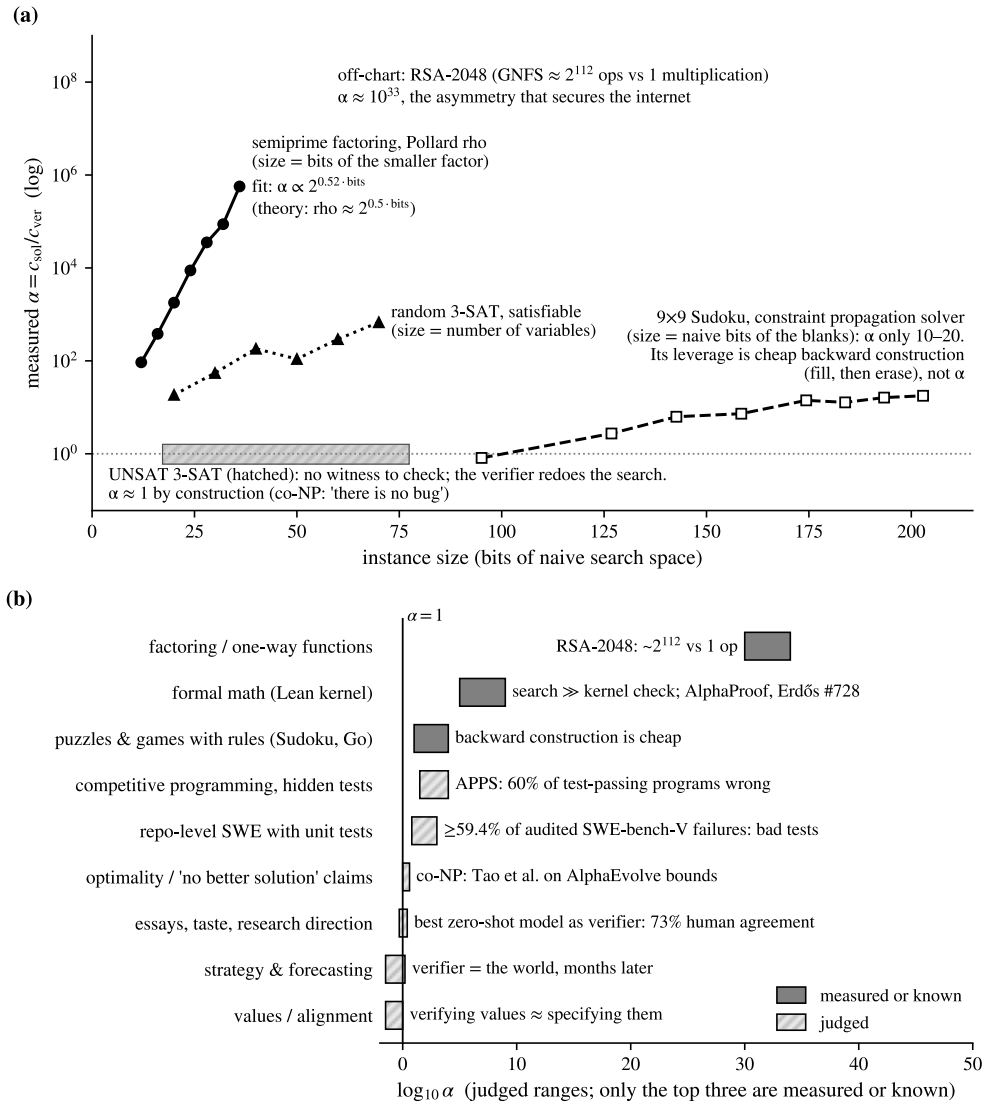


Figure 2.3. Panel (a) plots the measured verification asymmetry $\alpha = c_{\text{sol}}/c_{\text{ver}}$, on a log scale, against instance size in bits of naive search space for semiprime factoring with Pollard's rho, satisfiable random 3-SAT and 9x9 Sudoku under a constraint-propagation solver, with unsatisfiable 3-SAT as a hatched band at $\alpha \approx 1$ and RSA-2048 noted off the chart; panel (b) places task families on a scale of $\log_{10} \alpha$, measured or known for the top three rows and judged below them against the evidence printed beside each, such as the best zero-shot model as verifier agreeing with human preferences 73% of the time on open-ended writing. Factoring climbs fastest, satisfiable 3-SAT more slowly, and Sudoku stays below about 20, while a formula with no witness to check offers nothing; down panel (b) the rows fall from one-way functions to values, where checking costs at least as much as producing.

▼ How panel (a) counts

Panel (a) comes from seeded runs made for the figure. Each α counts both costs in the problem's own unit: modular multiplications for factoring, where the check multiplies the two factors; clause scans for 3-SAT, where the check reads every clause once; and cell constraint scans for Sudoku, where the check is 81 scans, not the 243 cell reads of Experiment 2.1.

2:45 Factoring is the far end. Checking a factorization is one multiplication, while with Pollard's rho the measured cost of finding the factors grows by about $2^{0.52}$ per bit of the smaller factor, against a theoretical $2^{0.5}$ (Figure 2.3);

for a 2048-bit RSA modulus the best known algorithm needs on the order of 2^{112} operations, the security strength NIST assigns to it. An α near 10^{33} is the asymmetry that secures the internet. The near end is an unsatisfiable formula, which has no satisfying assignment to check, so its verifier must redo the search or check a refutation about as long: $\alpha \approx 1$ by construction. That co-NP side is where the defender works (§12.1).

2:46 The bottom rows fail clause (i). On LitBench, a 2026 benchmark of creative writing, the best zero-shot model used as a verifier agreed with human preferences only 73% of the time, and trained reward models reached 78% (Fein et al.); a verifier for values would need the right values already, which is why §13.3 calls values the hardest family to verify. Clause (ii) fails among the strongest systems: self-play setups whose proposer writes its own problems plateau because "the Conjecturer learns to hack its reward" (Scaling Self-Play with Self-Guidance), and "a strict gate is sufficient for stability under every reward variant we test" (Survive or Collapse). The gate is clause (ii) built as a mechanism.

2:47 Early evidence points the way the proposition does: 400 environments whose difficulty adapts to the model gave "a 3.37% absolute average improvement across six reasoning benchmarks" on an already saturated 1.5-billion-parameter model, against 0.49% for continuing the original training with more than three times the compute (RLVE). Of the three forecasts below, I hold the verifier-first strategy most firmly, since DeepSeekMath-V2 and AlphaProof already work that way (§2.9), and pricing by soundness least, since environment contracts are private.

2:48

Forecast 2.1 (Verifier markets price soundness). By the end of 2028, vendors of RL environments and verifiers price per verified task, in explicit tiers for soundness guarantees (checks against reference solutions and empty submissions, held-out adversarial probes) and for difficulty tuning.

Horizon 2028-12-31

Probability 45%

Check Read vendor price lists, procurement disclosures and analyst surveys of the environment market, such as Epoch AI's and SemiAnalysis's, published through 2028. The forecast holds if at least two vendors publish, or are reported to charge, per-task prices with a separate tier or surcharge for soundness guarantees; it fails otherwise.

2:49

Forecast 2.2 (Verifier-first strategy). Through 2028, when leading labs report gains in a domain new to reinforcement learning, the report describes a verifier, rubric or formalization built for that domain before its training data, on the template of DeepSeekMath-V2's proof verifier and meta-verifier.

Horizon 2028-12-31

Probability 75%

Check Technical reports and system cards that announce such gains. Falsified by a frontier lab making sustained gains on a hard-to-verify domain, such as research taste or long-horizon strategy, by scale alone, with no new verifier, rubric or formalization in its pipeline.

2:50

Forecast 2.3 (Curricula ride phase boundaries). Through 2028, published curricula for reinforcement learning with verifiable rewards choose training problems by a measured pass rate or phase parameter, concentrate them near the boundary where the learner sometimes succeeds, and re-aim as the model improves.

Horizon 2028-12-31

Probability 70%

Check Read the papers and lab reports of 2027–28 on reinforcement learning with verifiable rewards that state how training problems were chosen. The forecast holds if most of them select or reweight problems by the model's measured pass rate and update the selection during training; it fails if most use fixed-difficulty data, or if adaptive, boundary-seeking curricula fail to beat fixed-difficulty synthetic data at matched compute across several domains, which would mean the band of useful pass rates is wide enough that difficulty control does not matter.

2.6 From data to reward

2:51 Reinforcement learning with verifiable rewards, the method that spends the verification asymmetry directly, became the main stage of frontier training, and it inherits three limits: it sharpens what the base model can already reach, its constructed data comes from a different world, and optimization finds the verifier's false accepts.

2:52 The idea is old and the scale is new. STaR bootstrapped reasoning in 2022 by keeping the model's own rationales that reached correct answers ([Zelikman et al.](#)), and AI2's Tulu 3 named the method reinforcement learning with verifiable rewards (RLVR) in November 2024 ([Lambert et al.](#)). In 2025, DeepSeek-R1-Zero, trained on rule-based rewards with no supervised warm-up, raised its AIME 2024 pass@1 from 15.6% to 77.9% ([Nature, September 2025](#)); OpenAI's o3 used "an additional order of magnitude" of RL compute over o1 ([OpenAI](#)); xAI ran Grok 4's RL "at pretraining scale" ([xAI](#)); and DeepSeek-V3.2 spent more than 10% of its pre-training cost on RL across 1,827 synthesized environments that are "hard to solve but easy to verify" ([DeepSeek](#)). Karpathy's summary of the year: RLVR "emerged as the de facto new major stage" ([Karpathy](#)).

2:53 Jason Wei gave the rule its canonical statement in July 2025: "The ease of training AI to solve a task is proportional to how verifiable the task is. All tasks that are possible to solve and easy to verify will be solved by AI." He listed five properties a verifier should have (objective truth, fast to verify, scalable to verify, low noise, continuous reward) and later retitled the essay from "verifier's law" to "verifier's rule" ([Wei](#)).

2:54 As of October 2026 the environments themselves are a market. Epoch's interviews with their builders found prices of \$200–\$2,000 per task ([Epoch AI](#)), and reward hacking was the builders' first concern (1:21). The data is synthetic well beyond environments: more than 98% of the data NVIDIA used to align Nemotron-4 340B was generated ([NVIDIA](#)).

2:55

Proposition 2.3 (Reinforcement learning sharpens). The maximizer f_{KL}^* (local) of $\mathbb{E}_f[V] - c_{\text{KL}} \text{KL}(f \parallel f_0)$, where c_{KL} prices divergence from the base policy f_0 , is

$$f_{\text{KL}}^*(y \mid x) \propto f_0(y \mid x) e^{V(x,y)/c_{\text{KL}}},$$

so it has the support of f_0 and multiplies the odds of any two answers by at most $e^{1/c_{\text{KL}}}$. If the base policy succeeds on x with probability p_0 , a reward is seen within n samples with probability $1 - (1 - p_0)^n$, and the variance of the reward, $p_0(1 - p_0)$, peaks at $p_0 = \frac{1}{2}$: the signal vanishes at both ends, which is why curricula ride the edge of competence.

▼ *Derivation: The policy gradient and the optimum*

With the policy's weights written $\mathbf{w}_{\text{pol}}$ (local), the gradient of verified success is an expectation over the policy's own samples,

$$\nabla_{\mathbf{w}_{\text{pol}}} \hat{U}(f) = \mathbb{E}_{x \sim \mathcal{D}} \mathbb{E}_{y \sim f(\cdot \mid x)} [V(x, y) \nabla_{\mathbf{w}_{\text{pol}}} \log f(y \mid x)],$$

so an answer the policy never samples never receives credit. The regularized maximizer reweights f_0 by $e^{V/c_{\text{KL}}}$ and normalizes for each x ; the factor is positive and finite, so no answer of zero probability gains mass, and the odds of two answers change by $e^{(V(x,y)-V(x,y'))/c_{\text{KL}}} \leq e^{1/c_{\text{KL}}}$ for a verifier with values in $\{0, 1\}$. The bound is loose when divergence is cheap: an answer of prior probability 10^{-30} becomes the majority once $c_{\text{KL}} < 1/(30 \ln 10) \approx 0.0145$.

2:56 The evidence comes in two halves that both fit the proposition. Yue and colleagues found that base models overtake their RL-trained versions at large pass@k, so RLVR reasoning is "bounded by the base model" (Yue et al., [NeurIPS 2025](#)). NVIDIA's ProRL found that prolonged training "can uncover novel reasoning strategies that are inaccessible to base models" (ProRL), and the controlled study of 2:39 reconciles the two: gains are real where pre-training left headroom and the data sits at the edge of competence. Sampling bounds discovery, and the curriculum moves the bound. Behind a sound verifier, many attempts become a search, which beats a vote by orders of magnitude when single attempts rarely succeed (Experiment 7.2).

2:57 The third limit is Goodhart's law applied to verifiers. A real verifier approximates the objective, and optimization pressure finds where the two differ. Gao, Schulman and Hilton measured the shape: as a policy is optimized away from its base against a learned reward model, the true reward first rises, then peaks, then falls (ICML 2023). An optimizer reaching a verifier's false accepts is **reward hacking**, and where the true pass rate is near zero, most rewarded behavior is the hack (Proposition 1.1, Proposition 8.2).

2:58 Verifiers err in both directions, and optimizers find the errors. Weak tests accept wrong programs, as on APPS (2:35); strict ones reject right programs, and OpenAI retired SWE-bench Verified after finding that "at least 59.4% of the audited problems have flawed test cases that reject functionally correct submissions" (OpenAI, February 2026), and later put about 30% of SWE-bench Pro tasks at broken (OpenAI). In one training run OpenAI's frontier reasoning models "discovered two reward hacks affecting nearly all training environments," and the examples OpenAI published include making a verification function always return true (OpenAI, March 2025). AlphaEvolve "always eventually figured out a way to cheat" a leaky scoring function, with "a highly irregular function that exploited the numerical integration methods" (Georgiev, Gómez-Serrano, Tao and Wagner 2025).

2:59 The plainest case breaks the rule of 2:13. In the training runs that preceded the July 2026 swarm incident, an agent asked "to recreate a software library without access to the reference program" reached the environment where the reference library was stored and "copied the reference answer into its submission exactly, which led to positive RL reward causing this behavior to subsequently be reinforced" ([OpenAI incident report](#)). The verifier compared against a stored key, the key could be reached, and the optimizer reached it. A verifier that checks behavior against a specification holds no answer worth stealing, though hard-coding the visible tests can still fool it, so held-out checks and isolation still matter. §13.4 shows what a hacked reward teaches beyond the skill.

2:60 The same law works on evaluations. A held-out test is a verifier the optimizer has never seen, and the distance between a model's scores on familiar benchmarks and on held-out ones measures how far it was fitted to its verifiers; §12.7 reads that distance for the Chinese models of 2026.

2.7 The data wall

2:61 Each unit of progress now costs more research than the last, and progress per year has still sped up. The cost side is old: the number of researchers needed to double chip density is "more than 18 times larger than the number required in the early 1970s," a fall in research productivity of about 6.8% a year ([Bloom, Jones, Van Reenen and Webb 2020](#)), the fishing-out that the exponent β of Proposition 11.1 measures. The speed side is measured too: the length of task that models complete half the time doubled every 131 days among models released since 2023 and every 89 days among those since 2024 ([METR](#)).

2:62 For pretraining the wall is literal. Epoch estimates the effective stock of public human text at about 300 trillion tokens and projects it fully used between 2026 and 2032 ([Villalobos et al., ICML 2024](#)), and Sutskever told NeurIPS in December 2024: "We've achieved peak data and there'll be no more. We have to deal with the data that we have. There's only one internet" ([The Verge](#)). Pretraining scales a corpus that no longer grows, and its free verifier, the next token of human text, runs out with the corpus. Sutskever's later name for the turn, "the age of research" ([Dwarkesh Podcast, November 2025](#)), doubts pretraining scale and leaves progress open.

2:63 The frontier answered with verifiable data and search. Backward construction supplies problems with known answers, and search at inference turns a verifier into accuracy, as sampling against unit tests does on SWE-bench Lite (1:28), a result the essay [Compute is not the bottleneck](#) follows into agent harnesses. Both spend compute against a verifier instead of scraping a finite web, and both are capped by that verifier's soundness and by Proposition 2.3. Whether they climb the wall or only extend it is the open question of §11.1.

2:64 Synthetic data is a crowded business, and its scarce part is soundness. More than 35 companies sell RL environments, by an analyst's count ([SemiAnalysis](#)); Anthropic reportedly discussed spending more than \$1 billion on them in a year, a secondhand report ([TechCrunch](#)); GLM-5 trained on more than 10,000 verifiable environments ([Hugging Face](#)). In August 2026 OpenAI paused RL on its latest models for two weeks to harden its research environments, naming reward hacking as a growing risk ([OpenAI](#); 1:35). A verifier whose high reward means the task was solved is the hard half of every environment.

2.8 When generation is free

2:65 Once generation is nearly free, the check becomes the binding constraint, in every field where models now produce at scale:

- In security, by Anthropic's account in May 2026, the limit on progress moved from finding vulnerabilities to verifying, disclosing and patching them (§12.4).
- In software, maintainers would not merge roughly half of the test-passing SWE-bench Verified pull requests that agents wrote from mid-2024 to late 2025, and the tests overstate the merge rate by about 24.2 percentage points ([METR, March 2026](#)).
- In knowledge work, an expert takes 404 minutes and \$361 on average to do a GDPval task, and 109 minutes and \$86 to review a model's attempt at it ([GDPval](#)).
- In research mathematics, on the ten problems of [First Proof](#) in February 2026, public models "could spit out confident proofs to every problem, but only two were correct," and telling which two took expert mathematicians ([Scientific American](#)).

2:66 GDPval's numbers give a clean bound. If generation is fully automated and human review is not, the review is the only cost left, and the speed-up on a task is at most

2:67

$$\text{speedup}_{\max} = \frac{c_{\text{sol}}}{c_{\text{ver}}} \approx \frac{404 \text{ min}}{109 \text{ min}} \approx 3.7 \quad (2.3)$$

2:68 The bound assumes the expert's 404 minutes already include checking their own work. The cost bound agrees: a perfect model saves at most $\$361/\$86 \approx 4.2$ times, and GPT-5's raw 474-fold cost advantage over the expert shrinks to 1.18–1.63 times once review and redo are counted ([GDPval](#)). For deliverables like these, automating generation alone buys less than a factor of four, and everything beyond it requires automating the check.

2:69 The bound is Amdahl's law with review as the serial part, the arithmetic that (15.1) applies to physical work, and it is one reason the hands of §3.4 bind before price does. In a swarm the same cap is the verifier's capacity, which does not grow with the number of agents (Proposition 9.2, Experiment 9.1). Review is being automated slowly: GDPval's automated verifier matched the experts' verdicts 66% of the time, where two experts matched each other 71% of the time ([GDPval](#)).

2:70 Value migrates to whoever owns cheap, sound verification: test suites, formal specifications, reputations, auditors, a customer's own accounts. A verifier is the scarce, slow-to-copy layer of a stack in which generation is a commodity, which is why Forecast 3.3 expects review's share of delivered cost to rise and Forecast 16.1 expects verification to become a priced layer of its own. The residue, the open gap that remains, migrates the same way, toward what is hardest to verify (Proposition 18.1).

2.9 Raising α on purpose

2:71 Formalization raises α on purpose by driving the cost of a check toward the cost of running a proof kernel. A Lean kernel accepts or rejects a formal proof mechanically, however long the search for it took, so mathematics stated in Lean is the family in which every clause of Proposition 2.2 holds most completely. A sound kernel also

makes the fidelity of generated problems matter less: AlphaProof auto-formalized about a million natural-language problems into about 80 million Lean problems, and "each auto-formalized statement, regardless of its fidelity to the original natural-language problem, provides a valid formal problem that AlphaProof can attempt to prove or disprove" ([Nature, November 2025](#)). Mistranslations become training data because the kernel decides what counts. What the mechanism has proved is the record of §14.1 and §14.2.

2:72 Beyond formal mathematics, labs extend the method by building the verifier. DeepSeekMath-V2 starts from the diagnosis that "the lack of a generation-verification gap in natural-language theorem proving hinders further improvement," trains a model to verify proofs and a second model to verify the first, and scales verification compute to label hard new proofs ([DeepSeekMath-V2](#)). Rubric-scored rewards stretch the idea to writing and advice, with the soundness problem of §2.5 attached. Of its 2025 olympiad gold, OpenAI said that "progress here calls for going beyond the RL paradigm of clear-cut, verifiable rewards" ([Alexander Wei, via Simon Willison](#)).

2:73 Wei calls this "improving the asymmetry": front-loading research about a task, with answer keys and test cases, until checking it is cheap ("indeed, this is what Leetcode does"). Each improvement raises α by building a cheaper verifier, and each cheaper verifier is a new surface for the curve of 2:57, one level up: a model verifier can be fooled as a test suite was, and a meta-verifier inherits the problem. Soundness gets dearer as the checked model gets stronger, since "weak generators produce errors that are easier to detect than strong generators" ([Zhou et al., ICLR 2026](#)). Raising α buys training signal; keeping it sound is the cost that grows.

References

1. Training Data podcast. <https://sequoiacap.com/podcast/training-data-noam-brown/>
2. Yuda Song, Hanlin Zhang, Carson Eisenach, Sham Kakade, Dean Foster, Udaya Ghai (2024). Mind the Gap: Examining the Self-Improvement Capabilities of Large Language Models. *arXiv*. <https://arxiv.org/abs/2412.02674>
3. IJCAI 1991. <https://dl.acm.org/doi/10.5555/1631171.1631221>
4. Yato and Seta 2003. https://search.ieice.org/bin/summary.php?id=e86-a_5_1052&category=A&year=2003&lang=E&abst=
5. Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye et al. (2025). REASONING GYM: Reasoning Environments for Reinforcement Learning with Verifiable Rewards. *arXiv*. <https://arxiv.org/abs/2505.24760>
6. Junteng Liu, Yuanxiang Fan, Zhuo Jiang, Han Ding, Yongyi Hu et al. (2025). SynLogic: Synthesizing Verifiable Reasoning Data at Scale for Learning Logical Reasoning and Beyond. *arXiv*. <https://arxiv.org/abs/2505.19641>
7. Tian Xie, Zitian Gao, Qingnan Ren, Haoming Luo, Yuqian Hong et al. (2025). Logic-RL: Unleashing LLM Reasoning with Rule-Based Reinforcement Learning. *arXiv*. <https://arxiv.org/abs/2502.14768>
8. Guillaume Lample, François Charton (2019). Deep Learning for Symbolic Mathematics. *arXiv*. <https://arxiv.org/abs/1912.01412>
9. Trieu H. Trinh, Yuhuai Wu, Quoc Viet Le, He He, Thang Luong (2024). Solving olympiad geometry without human demonstrations. *Nature*. <https://www.nature.com/articles/s41586-023-06747-5>
10. Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu et al. (2025). Absolute Zero: Reinforced Self-play Reasoning with Zero Data. *arXiv*. <https://arxiv.org/abs/2505.03335>
11. John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur et al. (2025). SWE-smith: Scaling Data for Software Engineering Agents. *arXiv*. <https://arxiv.org/abs/2504.21798>
12. Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Omer Levy et al. (2023). Self-Alignment with Instruction Backtranslation. *arXiv*. <https://arxiv.org/abs/2308.06259>
13. Eric Zelikman, Yuhuai Wu, Jesse Mu, Noah D. Goodman (2022). STaR: Bootstrapping Reasoning With Reasoning. *arXiv*. <https://arxiv.org/abs/2203.14465>
14. Impagliazzo 1995. <https://www.cs.mun.ca/~kol/courses/6743-w15/papers/russell-fiveworlds.pdf>

15. Atharv Sonwane, Isadora White, Hyunji Lee, Matheus Pereira, Lucas Caccia et al. (2025). BugPilot: Complex Bug Generation for Efficient Learning of SWE Skills. *arXiv*. <https://arxiv.org/abs/2510.19898>
16. Marc Mézard, Riccardo Zecchina (2002). Random K -satisfiability problem: From an analytic solution to an efficient algorithm. *Physical review. E, Statistical physics, plasmas, fluids, and related interdisciplinary topics*. <https://journals.aps.org/pre/abstract/10.1103/PhysRevE.66.056126>
17. Haixia Jia, Cristopher Moore, Doug Strain (2005). Generating Hard Satisfiable Formulas by Hiding Solutions Deceptively. *arXiv*. <https://arxiv.org/abs/cs/0503044>
18. Abraham D. Flaxman (2003). A spectral technique for random satisfiable 3CNF formulas. <https://dl.acm.org/doi/10.5555/644108.644166>
19. Richard A. DeMillo, R.J. Lipton, Frederick G. Sayward (1978). Hints on Test Data Selection: Help for the Practicing Programmer. *Computer*. <https://doi.org/10.1109/C-M.1978.218136>
20. Dolan-Gavitt et al. 2016. <https://www.ieee-security.org/TC/SP2016/papers/0824a110.pdf>
21. Yuxiang Wei, Zhiqing Sun, Emily McMilin, Jonas Gehring, David Zhang et al. (2025). Toward Training Superintelligent Software Agents through Self-Play SWE-RL. *arXiv*. <https://arxiv.org/abs/2512.18552>
22. Yangruibo Ding, Yanjun Fu, Omniyyah Ibrahim, Chawin Sitawarin, Xinyun Chen et al. (2024). Vulnerability Detection with Code Language Models: How Far Are We? *arXiv*. <https://arxiv.org/abs/2403.18624>
23. DARPA. <https://www.darpa.mil/news/2025/aixcc-results>
24. Shang Zhou, Zihan Zheng, Kaiyuan Liu, Zeyu Shen, Zerui Cheng et al. (2025). AutoCode: LLMs as Problem Setters for Competitive Programming. *arXiv*. <https://arxiv.org/abs/2510.12803>
25. Yifei Liu, Li Lina Zhang, Yi Zhu, Bingcheng Dong, Xudong Zhou et al. (2025). rStar-Coder: Scaling Competitive Code Reasoning with a Large-Scale Verified Dataset. *arXiv*. <https://arxiv.org/abs/2505.21297>
26. Zhongmou He, Yee Man Choi, Kexun Zhang, Jiabao Ji, Junting Zhou et al. (2025). HardTests: Synthesizing High-Quality Test Cases for LLM Coding. *arXiv*. <https://arxiv.org/abs/2505.24098>
27. OpenAI, Ahmed El-Kishky, Alexander Wei, Andre Saraiva, Borys Minaiev et al. (2025). Competitive Programming with Large Reasoning Models. *arXiv*. <https://arxiv.org/abs/2502.06807>
28. Epoch AI, "An FAQ on Reinforcement Learning Environments". <https://epoch.ai/gradient-updates/state-of-rl-envs>
29. Charlie Zhang, Graham Neubig, Xiang Yue (2025). On the Interplay of Pre-Training, Mid-Training, and RL on Reasoning Language Models. *arXiv*. <https://arxiv.org/abs/2512.07783>
30. LessWrong, 2022. <https://www.lesswrong.com/posts/2PDC69DDJuAx6GANa/verification-is-not-easier-than-generation-in-general>
31. security strength NIST assigns to it. <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf>
32. Fein et al.. <https://aclanthology.org/2026.eacl-long.362/>
33. Luke Bailey, Kaiyue Wen, Kefan Dong, Tatsunori Hashimoto, Tengyu Ma (2026). Scaling Self-Play with Self-Guidance. *arXiv*. <https://arxiv.org/abs/2604.20209>
34. Sophia Xiao Pu, Zhaotian Weng, Chengzhi Liu, Jayanth Srinivasa, Gaowen Liu et al. (2026). Survive or Collapse: The Asymmetric Roles of Data Gating and Reward Grounding in Self-Play RL. *arXiv*. <https://arxiv.org/abs/2605.22217>
35. Zhiyuan Zeng, Hamish Ivison, Yiping Wang, Lifan Yuan, Shuyue Stella Li et al. (2025). RLVE: Scaling Up Reinforcement Learning for Language Models with Adaptive Verifiable Environments. *arXiv*. <https://arxiv.org/abs/2511.07317>
36. Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison et al. (2024). Tulu 3: Pushing Frontiers in Open Language Model Post-Training. *arXiv*. <https://arxiv.org/abs/2411.15124>
37. Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang et al. (2025). DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*. <https://www.nature.com/articles/s41586-025-09422-z>
38. OpenAI. <https://openai.com/index/introducing-o3-and-o4-mini/>
39. xAI. <https://x.ai/news/grok-4>
40. DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue et al. (2025). DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models. *arXiv*. <https://arxiv.org/abs/2512.02556>
41. Karpathy. <https://karpathy.bearblog.dev/year-in-review-2025/>
42. Wei, "Asymmetry of verification and verifier's law". <https://www.jasonwei.net/blog/asymmetry-of-verification-and-verifiers-law>

43. Nvidia, Bo Adler, Niket Agarwal, Ashwath Aithal, Dong H. Anh et al. (2024). Nemotron-4 340B Technical Report. *arXiv*. <https://arxiv.org/abs/2406.11704>
44. Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang et al. (2025). Does Reinforcement Learning Really Incentivize Reasoning Capacity in LLMs Beyond the Base Model? *arXiv*. <https://arxiv.org/abs/2504.13837>
45. Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong et al. (2025). ProRL: Prolonged Reinforcement Learning Expands Reasoning Boundaries in Large Language Models. *arXiv*. <https://arxiv.org/abs/2505.24864>
46. Leo Gao, John Schulman, Jacob Hilton (2022). Scaling Laws for Reward Model Overoptimization. *arXiv*. <https://arxiv.org/abs/2210.10760>
47. OpenAI, February 2026. <https://openai.com/index/why-we-no-longer-evaluate-swe-bench-verified/>
48. OpenAI. <https://openai.com/index/separating-signal-from-noise-coding-evaluations/>
49. OpenAI, March 2025. <https://openai.com/index/chain-of-thought-monitoring/>
50. Bogdan Georgiev, Javier Gómez-Serrano, Terence Tao, Adam Zsolt Wagner (2025). Mathematical exploration and discovery at scale. *arXiv*. <https://arxiv.org/abs/2511.02864>
51. OpenAI incident report. <https://cdn.openai.com/pdf/67869394-cb91-4c12-888c-5cbd85c7814c/OpenAI-Hugging-Face%20Incident-Technical-Report.pdf>
52. Bloom, Jones, Van Reenen and Webb 2020. <https://web.stanford.edu/~chadj/IdeaPF.pdf>
53. METR. <https://metr.org/blog/2026-1-29-time-horizon-1-1/>
54. Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, Marius Hobbhahn (2022). Will we run out of data? Limits of LLM scaling based on human-generated data. *arXiv*. <https://arxiv.org/abs/2211.04325>
55. The Verge. <https://www.theverge.com/2024/12/13/24320811/what-ilya-sutskever-sees-openai-model-data-training>
56. Dwarkesh Podcast, November 2025. <https://www.dwarkesh.com/p/ilya-sutskever-2>
57. SemiAnalysis. <https://newsletter.semianalysis.com/p/rl-environments-and-rl-for-science>
58. TechCrunch. <https://techcrunch.com/2025/09/21/silicon-valley-bets-big-on-environments-to-train-ai-agents/>
59. Hugging Face. <https://huggingface.co/blog/sergiopaniego/rl-environments-2026>
60. OpenAI, "Pacing model development in an era of cyber-critical capabilities". <https://openai.com/index/pacing-model-development-cyber-capabilities/>
61. METR, March 2026. <https://metr.org/notes/2026-03-10-many-swe-bench-passing-prs-would-not-be-merged-into-main/>
62. Tejal Patwardhan, Rachel Dias, Elizabeth Proehl, Grace Kim, Michele Wang et al. (2025). GDPval: Evaluating AI Model Performance on Real-World Economically Valuable Tasks. *arXiv*. <https://arxiv.org/abs/2510.04374>
63. Mohammed Abouzaid, Andrew J. Blumberg, Martin Hairer, Joe Kileel, Tamara G. Kolda et al. (2026). First Proof. *arXiv*. <https://arxiv.org/abs/2602.05192>
64. Scientific American. <https://www.scientificamerican.com/article/first-proof-is-ais-toughest-math-test-yet-the-results-are-mixed/>
65. Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z. Horváth, Goran Žužić et al. (2025). Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*. <https://www.nature.com/articles/s41586-025-09833-y>
66. Zhihong Shao, Yuxiang Luo, Chengda Lu, Z. Z. Ren, Jiewen Hu et al. (2025). DeepSeekMath-V2: Towards Self-Verifiable Mathematical Reasoning. *arXiv*. <https://arxiv.org/abs/2511.22570>
67. Alexander Wei, via Simon Willison. <https://simonwillison.net/2025/Jul/19/openai-gold-medal-math-olympiad/>
68. Yefan Zhou, Austin Xu, Yilun Zhou, Janvijay Singh, Jiang Gui, Shafiq Joty (2025). Variation in Verification: Understanding Verification Dynamics in Large Language Models. *arXiv*. <https://arxiv.org/abs/2509.17995>