ARE HIDDEN CONSTANTS IN THEORETICAL COMPUTER SCIENCE REALLY THAT LARGE?

Anonymous authors
Paper under double-blind review

ABSTRACT

We use large language models to recover the hidden constants in theoretical computer science proofs, asking how large the constants are, why some become large, and how tighter analysis can improve the guarantees. We introduce **ConstAble**, which follows proof dependencies to derive explicit bounds and track their conditions. Our benchmark, **ConstEval**, evaluates bound recovery, proof validity, and error detection on 93 prepared instances and 93 paired variants with injected proof defects. Among 91 targets with numerical coefficients, 69 have representative coefficients below 1,000 under their stated resource models and parameter settings. The recovered proofs show how costs multiply across nested calls and why small leading coefficients can hide large costs at finite input sizes. The resulting bounds support comparisons at concrete input sizes, and our subsequent analysis tightens an earlier recovered three-distinctness bound without changing the algorithm. **ConstAble** improves bound completion over one-shot inference. With comparable median token use, it completes 65.6%–92.5% of targets, while baselines using additional generation or revision complete at most 9.68%.

1 INTRODUCTION

Algorithmic progress in theoretical computer science (TCS) is often measured by improvements in asymptotic complexity. Integer multiplication has advanced from $O(N^2)$ to $O(N \log N)$ time for N-bit integers (Karatsuba & Ofman, 1962; Schönhage & Strassen, 1971; Fürer, 2007; Harvey & van der Hoeven, 2021). Maximum flow has advanced from $O(nm^2)$ to almost-linear $m^{1+o(1)}$ time, where n and m count vertices and edges (Edmonds & Karp, 1972; Goldberg & Tarjan, 1988; Orlin, 2013; Chen et al., 2022). Yet better asymptotic bounds need not yield an advantage at realistic input sizes: *hidden constants* can delay that advantage dramatically. For example, Alman & Yu (2024) analyze standard recursive matrix multiplication with a leading factor $O(q^2)$ in its operation-count bound, where q is the base dimension. For a simplified Coppersmith–Winograd construction targeting exponent 2.5, their optimistic estimate gives $q \approx 1.33 \times 10^{56443}$, so even the base dimension far exceeds the roughly 10^{80} atoms in the observable universe. Algorithms whose asymptotic advantages emerge only at infeasible input sizes are known as *galactic algorithms* (Lipton, 2010). Such examples show that hidden constants can be enormous, but leave open what constants arise in other proofs and which steps determine their size. This raises our central question: *Are hidden constants in theoretical computer science really that large?*

Our survey of 75 papers reporting computational resource bounds from STOC, FOCS, and SODA found that only three (4%) made the constants explicit in all theorems corresponding to their main results.[1] Recovery requires more than filling in missing numbers: a lemma may be invoked at a different accuracy, a subroutine may be repeated inside another loop, and a bound may hold only above a threshold. Each affects the final coefficient or its validity range, and cited results can extend these dependencies across papers. A valid asymptotic argument is therefore not yet a complete explicit bound: its numerical costs and applicability conditions must be reconstructed and composed throughout the proof.

Our contributions. We use large language models (LLMs) to reconstruct the quantitative derivations left implicit in selected TCS proofs. We use the recovered derivations to measure coefficient sizes, explain how large factors arise, and identify estimates that can be tightened without changing

[1] Paper-level survey details, including the three qualifying papers, are provided in the supplementary material.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

the algorithm. We prepare self-contained benchmark documents with checked local and external premises and declared cost assumptions, including stipulated subroutine costs where needed. Models reconstruct the remaining internal calculations and compose explicit bounds under these assumptions. Our contributions are threefold:

- **ConstEval: a benchmark for hidden-constant recovery.** We construct 93 recovery instances from 26 textbook expositions and 67 curated research-paper documents, spanning algorithms, complexity theory, and quantum computation. Each has a paired variant with injected proof defects, giving 186 instances in total (Table 3). We assess bound completion, proof validity, and defect detection separately: completion requires explicit bounds and supporting derivations for every requested clause and parameter regime. Separate model checks of proof correctness and quantitative expansion are supplemented by expert spot checks and adjudication (Section 4.1).
- **ConstAble: a pipeline guided by proof dependencies. ConstAble** traces the proof route, builds a dependency graph, and expands extracted local arguments through review and revision. Each node retains its explicit bound, applicability conditions, and supporting derivation for downstream composition (Section 2.2). Across four paired model–execution settings, bound completion rises from 1.08%–9.68% to 58.1%–92.5%, with higher injected-defect coverage under the respective inference budgets. Across all 67 research targets, the pooled substantive-error rate falls from 9.0% to 3.4% (Table 1). In the API settings, the completion advantage persists against repeated-generation and review–revision baselines with approximately comparable pooled median token use (Table 2).
- **What recovered constants reveal: magnitude, amplification, and finite-size guarantees.** Numerical coefficients are available for 91 of the 93 targets; 69 (76%) are below 1,000, including 48 of 65 research targets (74%) and 21 of 26 textbook targets (81%). The largest coefficient reaches 1.27×10^{39} at its stated parameter setting (Section 3.1). Cost traces locate the steps where nested iteration and enumeration amplify factors. In the quantum-money case, each sample's validation cost also scales with the sample count, so the sampling coefficient is squared (Section 3.2). The complete resource-specific bounds support comparisons at concrete input sizes: restoring the quantum query coefficient moves its intersection with the classical lower bound from 16 to approximately 265 (Section 3.4).

Together, these contributions turn quantitative reconstruction into a tool for studying algorithmic guarantees and benchmarking LLM reasoning. Recovered derivations explain numerical costs and expose opportunities to tighten proof estimates. **ConstEval** tests whether models can reconstruct omitted calculations, preserve conditions across dependencies, and distinguish supported conclusions from defective arguments. **ConstAble** provides a structured approach with inspectable intermediate proofs. By separately measuring completion, proof validity, and defect detection, this setting assesses the ability to assemble complete quantitative arguments whose results inform finite-size analysis.

Related work. Language-model theorem proving includes GPT-f in Metamath (Polu & Sutskever, 2020) and systems using Lean (de Moura & Ullrich, 2021). LeanDojo combines premise retrieval with proof generation in ReProver (Yang et al., 2023). ProcessBench evaluates whether models can locate the earliest erroneous step in a mathematical solution or identify a fully correct solution (Zheng et al., 2025). **ConstEval** instead evaluates quantitative reconstruction from supplied TCS proofs, requiring explicit resource bounds with supporting derivations and testing whether defects affecting recovery are identified or corrected.

At the method level, proof dependencies and their extraction have been studied in interactive theorem proving and formal mathematical libraries (Bertot et al., 2000; Alama et al., 2012). DeepSeek-Prover-V2 uses recursive subgoal decomposition to construct formal proofs for training (Ren et al., 2025), while Self-Refine iteratively revises outputs using feedback from the generating model (Madaan et al., 2023). **ConstAble** adapts these ideas to recovering and composing quantitative bounds along an existing proof, with local review tracking numerical costs and applicability conditions. Unsupported premises remain explicit obligations.

Automatic resource analysis derives quantitative bounds from programs and structured cost models (Albert et al., 2007; Hoffmann & Hofmann, 2010); proof mining extracts effective bounds and computational information from mathematical proofs (Kohlenbach, 2008). We recover explicit resource bounds from TCS expositions using LLMs, with model checks and expert audits but no formal certification. Appendix A expands these comparisons.

2 TASK, DATASET, AND RECOVERY PIPELINE

We test whether an agent can turn a qualitative resource bound in a supplied TCS proof into a checkable numerical bound, carrying the costs and conditions of invoked results into the target. An instance supplies a LaTeX proof document and a target specification: quantity, parameter domain, charged resource, guarantee mode (such as worst-case or expected cost), and result identifier. A target may have several clauses or regimes. For $T_{\mathrm{res}}(x) = O(g(x))$, recovery requires an explicit $C > 0$ and a derivation showing

$$T_{\mathrm{res}}(x) \le Cg(x) \qquad \text{for every } x \in \mathcal{D}, \tag{1}$$

where g is the target's fixed growth term, $\mathcal{D}$ covers the target's original validity domain without extra restrictions, and C is independent of varying parameters x. The answer states any fixed structural parameters on which C depends. The same requirement applies to requested lower bounds, exponent constants, lower-order terms, probability losses, and validity thresholds. Every clause needs calculations through invoked source lemmas; a named but unresolved coefficient is incomplete. Alternative valid derivations within the supplied proof strategy are accepted unless a canonical expression or smallest integer constant is prescribed. Unsupported premises and their affected conclusions must be identified. Completion concerns this proof obligation; numerical tightness is compared separately on common admissible inputs.

2.1 CONSTEVAL: PREPARED PROOFS AND PAIRED DEFECTS

ConstEval has 93 positive documents: 26 constructed textbook expositions and 67 curated research documents (Appendix Table 3). Research selection starts from 100 papers across algorithms (34), complexity (33), and quantum computation (33). A retained target has its full proof in the PDF and at most three distinct substantive external results across traced dependencies. This permits manual verification while retaining chained, branching, and shared dependencies. Tasks span running time, space, query complexity, circuit size, and lower bounds. In the initial research pool, 69 of 100 targets invoke a substantive external result.

Preparation makes quantitative premises available. GPT-5.6 Sol assists retrieval; human reviewers verify source versions, applicable statements, and premises, with further GPT-5 Pro retrieval and human checks for unresolved cases. Preparation clarifies ambiguous internal statements, supplies checked external results, and declares primitive-operation or subroutine costs under a reference model. Source expansions remain visible; the solver reconstructs calculations left unresolved by the prepared document. A stipulated cost is an assumption that may be composed arithmetically, not a cost established by recovery. A separate checker reconstructs the target with the reference answer and construction reasoning withheld, checking ranges, boundary cases, dependencies, and answer leakage. Unresolved specification or proof gaps exclude a positive entry. Appendix B.1 details construction and gives quantitative examples.

Each positive document has a paired negative document that changes its supporting proof while retaining the final statement, proof route, and valid LaTeX. The 93 negatives contain 70 one-block and 23 two-block modifications: 116 faulty blocks and 147 annotated atomic errors. Each block has one to three errors; paired blocks are independently diagnosable and each undermines the target argument. Annotations give the source anchor, consequence, and minimal repair. A negative label concerns the derivation; the theorem may still hold. Fault types and examples appear in Appendix B.1.

2.2 CONSTABLE: RECOVERY THROUGH DEPENDENCIES

ConstAble follows four reviewed stages (Figure 1). The roadmap locates the target, regimes, and proof route. Graph construction follows invoked results to atomic nodes, recording definitions and setup separately; atomic nodes may still hide coefficients. Proof extraction supplies a local derivation for each node, treating direct dependencies as premises while retaining assumptions, substitutions, inequalities, and cases. Missing local support stays visible.

Resolution works from dependencies toward the target. For each node, it makes coefficients, thresholds, absorption, and probability accounting explicit; substitutes dependency parameters; and combines applicability conditions. The updated node record retains an explicit bound, conditions, derivation, and source anchor. Supplied external bounds and stipulated costs enter with their conditions;

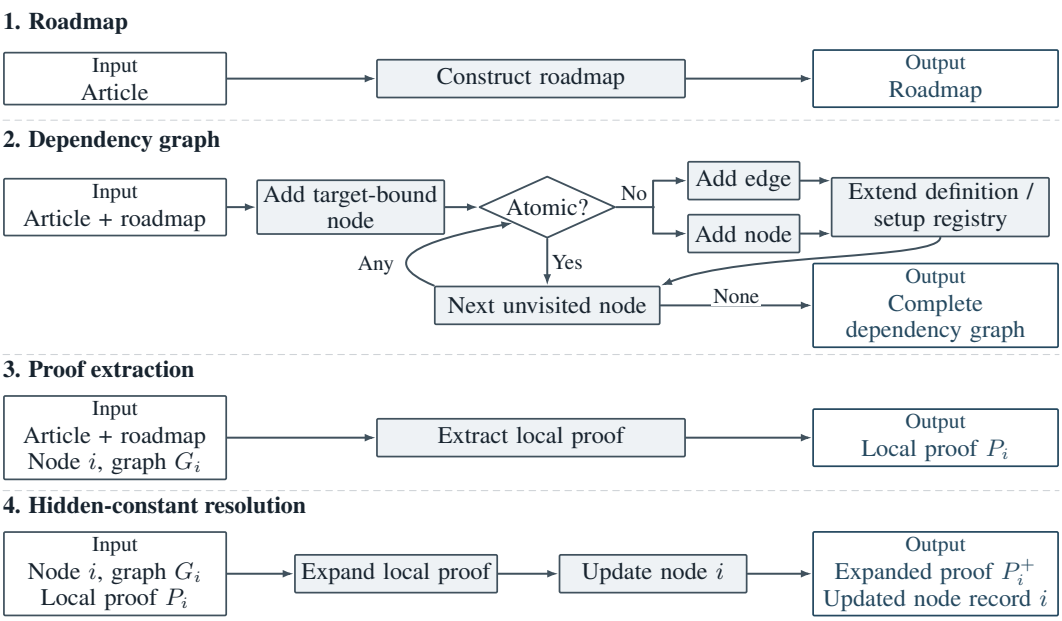


Figure 1: ConstAble's four recovery stages. The article is paired with the target specification in Section 2. The roadmap locates the target and proof route. Graph construction traces dependencies to atomic nodes and records definitions and setup separately. For each node i requiring quantitative resolution, the article, roadmap, and local graph G_i yield a proof P_i. Resolution expands it to P_i^+ and updates the node's bound for use by its dependents.

unsupported premises remain open obligations. The final output collects the target bound and calculations supporting every invoked coefficient. Stage reviews check target coverage, graph completeness, extraction faithfulness, and numerical consistency. Independent proof and expansion checks used for evaluation appear in Section 4.1; full stage and prompt details appear in Appendices B.3 and G.3.

3 What the Recovered Constants Reveal

Among 65 research targets with numerical coefficients, 48 have a representative coefficient below 1,000; across all 91 numerical targets, 69 do. At the other extreme, a geometric-matching coefficient reaches 1.27×10^{39}. These values are not interchangeable estimates of practical cost. Their meaning depends on the charged resource, growth term, parameter setting, and domain. We first explain what the distribution measures, then trace two ways proof structure changes a coefficient, and finally show what explicit bounds permit at finite input sizes.

3.1 The Distribution and Its Units

The numerical inventory contains coefficients for 91 of 93 targets: 26 textbook targets and 65 research targets, comprising 27 algorithmic, 16 complexity-theoretic, and 22 quantum results. We summarize their magnitudes under the stated resource units and parameter settings. Appendix C.1 describes the analysis grouping and coefficient-extraction rules, and Table 7 lists the complete inventory.

Twenty-one of 26 textbook coefficients and 48 of 65 research coefficients lie below 1,000, giving 69/91 (about 76%) overall. Eight lie in $[10^3, 10^5)$ and 14 are at least 10^5. Figure 2 displays 90 entries on a log scale, showing reciprocal markers for lower-bound coefficients. The omitted [32] entry, $1/9216$, is an exponent saving in a rank upper bound; displaying it as an ordinary multiplicative cost would obscure its role.

Moderate values and the upper tail. Textbook examples include Floyd–Warshall's 123 in an n^3 step bound and Euclid's 24 in a logarithmic bound. Twenty of 27 research-level algorithm coefficients are also below 1,000: Eulerian-cycle construction has 52 in a step bound, and fair allocation has

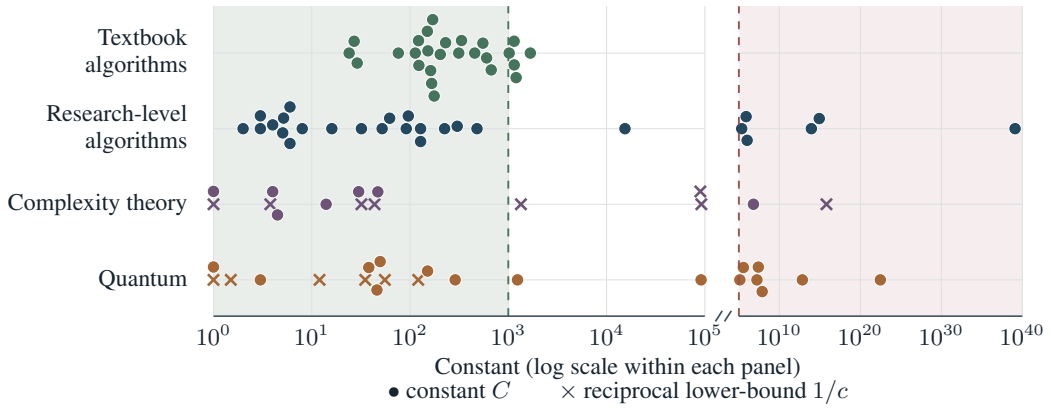


Figure 2: Recovered coefficients across four TCS groups. Dots show constants; crosses show reciprocals of lower-bound constants. Vertical offsets separate nearby points. Green shading marks values below 10^3, red shading values at least 10^5.

$81/16$ in a comparison bound. Fifteen of 16 complexity coefficients lie below 1,000. Here the direction of a guarantee matters: [25] proves at least $2^{0.264n}$ binary $+$ gates for its matching support function at $n \geq 6$, whereas [37] has an exponent coefficient near 1.53×10^{-16} only on a specified infinite family of Boolean ranks. Thirteen of 22 quantum coefficients are below 1,000; finding all 1s has coefficient $(\pi + 5)/2$ in a query bound of order $\sqrt{N\bar{k}}$, while its representative gate coefficient is $178 + 35\pi$. The tail reaches 2.94×10^{22} for nested-update three-distinctness [56] and 1.27×10^{39} for geometric matching [08] at $\omega = 3$. The tail therefore crosses research areas, while its numerical scale changes with each target's resource and normalization.

The proof-graph analysis offers a useful qualification. Across all 91 targets, associations between coefficient rank and graph size or shape are positive but modest; the node correlation is only 0.051 within research papers, compared with 0.578 among textbook targets. Paper length has no resolved association. Appendix C.2 gives the complete correlation table, intervals, multiplicity adjustment, and sensitivity analysis. These associations do not identify a causal source of large coefficients; the cost traces below show where particular factors enter.

3.2 How Large Factors Arise

A shared cost can be multiplied by itself. In quantum money [52], the sample count s and the validation cost per sample both depend on the same accuracy calculation. With failure budget f, the latter is proportional to s/f, so the total contains s^2/f. Its sample coefficient

$$A = \frac{18432(\ln 6 + 1)}{\pi^2} \simeq 5.21 \times 10^3 \quad \text{becomes} \quad C = \frac{9\pi^2}{32} A^2 \simeq 7.55 \times 10^7$$

in the validation bound. Keeping its growth term and conditions gives

$$t \leq C f^{-1} \varepsilon^{-4} \ln^2(1/\eta), \quad 0 < f, \varepsilon \leq 1, \quad 0 < \eta \leq e^{-1}.$$

Here t counts validations of a qubit state. The guarantee has two levels: failure probability at most f, and, conditioned on no failure, probability at least $1 - \eta$ of fidelity at least $1 - \varepsilon$. Thus reducing the admissible infidelity affects the sample count and then the validation budget; the recovered fourth power of $1/\varepsilon$ and the squared numerical coefficient expose that same composition. The square is a consequence of two cost-bearing uses of s, not merely a long proof. A distinct nested-update three-distinctness target [56] compounds attempts and inner search, producing 2.94×10^{22} against $n_0^{5/7} \log_2^6 n_0$; it should not be confused with the [62] refinement below. Geometric matching [08] compounds a $2^{24\omega}$ factor with repeated inverse powers of $1 - (15/16)^{\omega/2-1}$, yielding its extreme value at $\omega = 3$. Appendix C.3 shows the ledgers and further examples.

Enumeration can be large even when the recorded leading coefficient is small. For BINARY-CSP [23], a domain of size q has $M = 2^q - 1$ nonempty masks. A merge involving at most d blocks

can enumerate M^{d+1} tuples. At $q = 8, d = 3$, this is $255^4 \simeq 4.23 \times 10^9$ candidates per merge. That illustrative setting differs from the inventory's $q = 2, d = 1$ setting, where the reported leading coefficient is $31/6 \simeq 5.167$ relative to $M^d n^2$. More explicitly, the ledger contains

$$T_{\text{enum}} = 66(n-1)M^{d+1}\big((d+1)^2q^2 + d + 1\big),$$

in addition to a quadratic term $(31/2)n^2$ and the remaining bookkeeping costs. With q, d fixed, the enumeration is linear in n and disappears from the leading-term summary, even though its coefficient can be large. Increasing q or d can simultaneously enlarge this enumeration term and decrease the reported coefficient $31/(2M^d)$ because the normalization also changes. Thus the small leading entry and the large enumeration cost are compatible facts about one proof. This is why the ledger, rather than only the leading coefficient, is needed for a finite-size comparison. Accuracy allocation creates another factor: dynamic volume estimation [10] uses internal accuracy $\varepsilon/3$, multiplying an inverse-square sampling term by nine; its reported update coefficient is 223450. These examples make the full expression, not an isolated coefficient, the object needed for finite-size decisions.

The same distinction holds even without a change in parameters. For quotable signatures [04], the representative coefficient of the cubic term is only 3, yet the recovered branches include linear coefficients above 268 million and additive terms above 134 million. Calling this a "small-constant" construction from its leading entry would discard precisely the terms that can dominate at moderate B, the supplied input and adversary cost scale. In data-oblivious graph algorithms [17], the selected I/O coefficient also reflects a conversion between cost normalizations and applies under stated density thresholds. A large entry can therefore result from resource accounting as well as repeated algorithmic work. The appendix retains the full ledgers and parameter conditions for both examples.

3.3 When a Recovered Guarantee Can Be Tightened

For prior-knowledge three-distinctness [62], an earlier recovered proof used one coefficient of about 2.95×10^{20} for both input-size regimes. Re-examining that proof after recovery, using its supplied multiplicity estimates and learning-graph conversion, gives $50331648\, n^{5/7}$ queries for $n \geq B = 2^{68}$. For $k = 3$ and $n \geq 3$, the supplied nonnegative multiplicity estimates satisfy $|\widehat{T}_t - T_t(x)| \leq \lceil n^{1/4} \rceil$ for $t = 1, 2$. Only original-input-oracle queries are charged, with error at most $1/4$. The two available branches are

$$Q(n) \leq \begin{cases} n, & 3 \leq n < B, \\ 50331648\, n^{5/7}, & n \geq B. \end{cases}$$

The earlier common coefficient used $B \simeq 2.95 \times 10^{20}$ to absorb the scan. This is sufficient but loses the scaling already present in $n^{5/7}$. The decisive normalization is

$$\frac{n}{n^{5/7}} = n^{2/7} < B^{2/7} = (2^{68})^{2/7} < 50331648 \qquad (n < B).$$

Thus the large-input coefficient also covers the scan, reducing that earlier proof's common coefficient to 5.03×10^7 under the same algorithm, assumptions, and success guarantee. This is a post hoc author calculation, separate from model output. It is also separate from the current [62] inventory entry 2.54×10^7, which came from a different, sharper derivation; 5.03×10^7 is not claimed to improve the inventory bound. The improvement is to this proof's guarantee, not a measured speedup or a new best constant for the original algorithm. The extended analysis accompanying the applications below distinguishes these derivations.

3.4 What Explicit Constants Make Possible

Explicit constants support three uses beyond asymptotic classification: comparing finite-size guarantees, tightening proof estimates, and translating accuracy or success requirements into resource budgets. Appendix C.4 collects the calculations, the comparison of refinement derivations, and resource-planning examples.

Finite-size comparisons. Explicit coefficients determine where competing bounds intersect. For finding all 1s [48] in an N-bit string with at most sixteen marked bits, the exact quantum algorithm has query upper bound $4.0708\sqrt{16N}$, whereas the exact classical lower bound is N. Setting the coefficient to one places their intersection at 16; restoring it moves the intersection to about 265.

Among the construction's power-of-two input lengths, 512 is therefore the first size at which these bounds certify a quantum query advantage. The appendix also compares Dijkstra implementations, where graph density changes which recovered upper bound is smaller.

Sharper guarantees. Section 3.3 shows how normalizing a small-input branch by the target growth term tightens an earlier recovered guarantee without changing the algorithm.

Applications and resource planning. Beyond these recovered examples, explicit bounds support resource decisions in several domains. In *quantum computation and scientific simulation*, constants in approximation bounds determine sufficient circuit-segment and logical-gate budgets for a target accuracy; translating these budgets into qubit and runtime estimates additionally requires hardware and error-correction assumptions. In *cryptography*, explicit reduction losses quantify how a primitive's security guarantee transfers to a protocol. In *statistical learning*, numerical error bounds determine sufficient sample sizes: an error bound $C/\sqrt{n}$ is at most ε whenever $n \geq (C/\varepsilon)^2$. In *streaming algorithms and data structures*, explicit coefficients translate accuracy and failure-probability requirements into storage and update budgets. The appendix illustrates this accounting with quantum simulation, signature security, PAC-Bayes bounds, and Count-Min sketches.

4 Evaluation of Quantitative Recovery

We assess whether answers supply complete, checkable quantitative proofs and whether additional generation achieves the same result. Proof validity and injected-defect detection are measured separately from bound completion.

4.1 What Counts as Recovery?

ConstEval has 93 positive targets and 93 paired negative documents. Each negative document retains its target and proof route but changes one or two supporting argument blocks; the collection contains 147 annotated atomic errors. A positive target counts as complete only when every requested clause and parameter regime has an explicit bound supported by a correct, checkable derivation, including constants in invoked source lemmas. A valid weaker or conditional bound need not complete the requested target. Unfinished runs count as failures in the 93-target completion rate R_{comp}. Negative-instance coverage R_{cov} counts distinct correctly identified annotations out of 147; it measures detection of the injected defects, not completion of the original bound.

Each delivered answer receives separate GPT-6 Sol checks of proof validity and quantitative expansion, using agent instances distinct from answer generation. TCS experts audit a uniformly random sample of approximately one fifth of the evaluation data and one or two short GPT-5.6 Sol proof fragments per article. In the reported proof-check audit, they overturn 23 of 381 judgments; GPT-5 Pro revises those judgments in at most two rounds, and the experts accept the revisions. These are adjudications of sampled checking judgments, not a claim that experts independently verified every final proof. Submitted answers remain fixed. A separate GPT-6 Sol check matches defect detections to the frozen annotations, with expert confirmation of all sampled decisions in a random 10% audit. Appendix D.1 gives the complete grading rules.

Codex runs use an agent environment with file and tool access, whereas API runs use the pipeline implementation; the modes therefore have different interfaces and inference settings. The paired comparisons below hold the target files and model within each execution mode fixed. DeepSeek API and one-shot experiments use version `0731`. Six Codex configurations and their different reasoning or runtime allowances are in Appendix D.2; their completion and detection profiles appear in Fig. 5.

4.2 Can the Pipeline Complete Checkable Bounds?

Completion and defect detection. Across the four paired Qwen3.7 Plus and DeepSeek v4 Flash settings in Table 1, one-shot completion is 1.08%–9.68%; ConstAble reaches 58.1%–92.5%. The gains occur in both Codex and API execution, though the within-mode comparisons retain their respective inference budgets. Annotation coverage also improves in each pair, by 25.8–39.4 percentage points. These are two different achievements: a method can identify a faulty local step without reconstructing every constant in the positive target.

Table 1: Workflow comparison at xhigh effort under the evaluated budgets. Completion uses positive instances 1–93. Coverage uses 147 negative error annotations. Substantive-error rates use all 67 research-paper positive instances as their denominator. Bold marks the better value in each one-shot/pipeline pair for completion, coverage, and error rate.

	Qwen3.7 Plus (xhigh)				DeepSeek v4 Flash (xhigh)			
	Codex		API		Codex		API	
Metric	One-shot	Pipeline	One-shot	Pipeline	One-shot	Pipeline	One-shot	Pipeline
Bound completion R_{comp} (%)	3.23	**58.1**	1.08	**65.6**	9.68	**83.9**	5.38	**92.5**
Annotation coverage R_{cov} (%)	53.1	**85.7**	49.0	**88.4**	62.6	**88.4**	55.8	**83.0**
Substantive errors	12	4	9	4	2	1	1	0
Error rate R_{err} (%)	17.9	**6.0**	13.4	**6.0**	3.0	**1.5**	1.5	**0.0**

Validity of delivered proofs. The proof audit covers the same 67 research articles in each of the four model–execution settings per workflow: 268 article–configuration outputs per workflow. A substantive error is a false or unsupported essential step in the asserted conclusion, including its parameter range, resource accounting, or probability guarantee. A local numerical error can be corrected without changing the justified argument. On this fixed denominator, the adjudicated substantive-error count falls from 24/268 (9.0%) for one-shot to 9/268 (3.4%) for ConstAble.

In the circuit-energy target [19], DeepSeek Codex recovers $E_{\text{circ}}(f) \leq 14D_{\text{tree}}(f)^3$, while Qwen Codex leaves a connector gate-count coefficient unresolved. Under API execution, Qwen recovers 14 and DeepSeek leaves that coefficient open. Circuit energy counts gates outputting one; D_{tree} is decision-tree depth. The unresolved local calculation prevents completion even when the target's growth order is retained (Appendix D.2).

4.3 Is More Generation Enough?

The API controls spend roughly comparable inference in three other ways: 13 independent one-shot generations, four run–review–revise rounds (12 calls), or 13 sequential refinements. Token use is uncached input plus output tokens, summed across calls. Median instance-level ratios to each model's one-shot run are measured over the pooled 93 positive and 93 negative instances, not separately for positive targets. The pipeline ratios are $11.12\times$ for DeepSeek and $12.69\times$ for Qwen; expanded-control ratios span $10.57\times$–$15.87\times$. Fixed call counts were calibrated to these pooled medians, not to equal token use on every instance. Table 2 gives the per-strategy token ratios.

Table 2 shows that no expanded control exceeds 9.68% completion, against 92.5% for DeepSeek's pipeline and 65.6% for Qwen's. Parallel sampling receives credit if any of its 13 independently checked answers completes a target, yet reaches only 8.60% and 3.23%. This retrospective any-success score does not provide an automatic selector for the successful answer. This supports a narrower conclusion than a strict equal-budget advantage: allocating similar pooled median tokens to additional whole-answer attempts does not reproduce dependency-wise expansion under these protocols. On negative documents, the union across 13 independent answers finds 81.6% of DeepSeek's and 82.3% of Qwen's annotated errors, close to pipeline coverage of 83.0% and 88.4%. Repeated generation thus broadens local defect discovery much more than it completes all quantitative obligations in one proof.

The component variants help locate the bottleneck but use their own inference allocations, so their differences are not isolated cost-controlled effects. Replacing incremental dependency construction with one graph call reduces Qwen API completion from 65.6% to 47.3%, the largest reported ablation gap. Removing the roadmap, proof extraction, or review–revision gives 57.0%, 54.8%, or 60.2%. In a separate comparison with graph review disabled in both variants, incremental construction reaches a final output on 87/93 tasks versus 63/93 for single-call construction. That is a run-progress measure, not the fraction with a complete, correct bound. The paired counts and failure traces are in Appendix E.4.

Table 2: Completion (%) and median token ratios for expanded inference (left), with component ablations (right). All configurations use API calls. Token ratios pool positives and negatives and measure uncached tokens relative to the same model's one-shot run (Appendix E.1). Parallel sampling counts any complete answer among 13. Ablation token ratios are not reported.

(a) Expanded inference.

	DeepSeek v4 Flash		Qwen3.7 Plus	
Configuration	Complete	Tokens	Complete	Tokens
Full pipeline	**92.5**	11.12×	**65.6**	12.69×
Parallel sampling (13 calls)	8.60	12.46×	3.23	12.64×
Review–revise (12 calls)	9.68	10.57×	0.00	11.90×
Iterative refinement (13 calls)	6.45	11.64×	1.08	15.87×

(b) Component ablations.

	Qwen3.7 Plus
Configuration	Complete
Without review–revision	60.2
Single-call	47.3
Without roadmap	57.0
Without proof extraction	54.8

Finally, completion and numerical tightness answer different questions. A completed explicit upper bound can be conservative, while a smaller unsupported coefficient is not a valid recovery. Appendix D.4 compares accepted bounds only where resource, unit, and admissible regime agree; Appendix D.3 gives a case where a sound parameter-dependent result remains weaker than the target. These limits matter when interpreting the completed coefficients in Section 3.

5 Conclusion, Limitations and Future Work

We presented **ConstAble**, which reconstructs explicit resource bounds by composing local quantitative proofs and their applicability conditions, and **ConstEval**, which separately evaluates bound completion, proof validity, and defect detection. Under the evaluated budgets, ConstAble improves completion over one-shot inference, with fewer substantive proof errors in the pooled audit; the completion advantage persists against expanded-inference baselines in the API experiments. Among the 65 research targets with numerical coefficients, 48 are below 1,000 under their stated normalizations and parameter settings, while the largest reaches 1.27×10^{39}. Cost traces identify how nested iteration, enumeration, and accuracy allocation amplify numerical factors. The resulting explicit bounds support finite-size comparisons and reveal proof estimates that can be tightened without changing the underlying algorithm.

The main benchmark uses a selected corpus with manually prepared premises and limited external dependencies, so its recovery rates and coefficient distribution describe this prepared setting rather than arbitrary TCS papers. Evaluation combines model checks and expert audits without formal certification; separate agent instances may share errors, and defect coverage does not measure false alarms. Expanded baselines approximately align median token-use ratios, without per-instance budget matching. Recovered coefficients also depend on the specified growth terms, resource models, parameter settings, and proof estimates, and need not be optimal.

A next step is to scale cross-paper recovery beyond the current case studies (Appendix F) and build reusable libraries of explicit bounds, retaining their sources, resource units, applicability conditions, and supporting derivations. Machine-checkable local proofs and verified composition could strengthen the reliability of these records. Beyond recovery, cost traces suggest a route to systematic bound refinement: identify estimates that contribute the most slack, derive tighter replacements under the same assumptions, and propagate their effects through dependent results. Implementations aligned with the declared resource models could further assess how informative the resulting finite-size guarantees are for algorithm selection and resource planning.

AI USE STATEMENT

We used large language models to assist with source retrieval, preparation of quantitative premises and reference derivations, bound recovery, and evaluation. Generative AI also assisted with manuscript editing. The authors take responsibility for the paper's claims, text, and artifacts.

ETHICS STATEMENT

This study involves no human-subject experiments or sensitive data. It evaluates mathematical reasoning on constructed textbook expositions and curated research proofs. The benchmark's negative instances contain deliberately introduced proof defects, documented through their source locations, consequences, and repairs.

REPRODUCIBILITY STATEMENT

We will publicly release the code, benchmark data, and prompts. Section 2.1 describes data preparation and defect annotation; Section 2.2 specifies the recovery workflow. Evaluation criteria and adjudication appear in Section 4.1. The appendices describe the available inference protocols, measured API token consumption, prompts, and evaluation criteria.

REFERENCES

Jesse Alama, Lionel Mamane, and Josef Urban. Dependencies in formal mathematics: Applications and extraction for Coq and Mizar. In *Intelligent Computer Mathematics*, volume 7362 of *Lecture Notes in Computer Science*, pp. 1–16. Springer, 2012. doi: 10.1007/978-3-642-31374-5_1. URL https://arxiv.org/abs/1109.3687.

Elvira Albert, Puri Arenas, Samir Genaim, Germán Puebla, and Damiano Zanardini. Cost analysis of Java bytecode. In *Programming Languages and Systems: 16th European Symposium on Programming (ESOP)*, volume 4421 of *Lecture Notes in Computer Science*, pp. 157–172. Springer, 2007. doi: 10.1007/978-3-540-71316-6_12. URL https://doi.org/10.1007/978-3-540-71316-6_12.

Josh Alman and Hantao Yu. Improving the leading constant of matrix multiplication. *arXiv preprint arXiv:2410.20538*, 2024. doi: 10.48550/arXiv.2410.20538. URL https://arxiv.org/abs/2410.20538.

Mihir Bellare and Phillip Rogaway. The exact security of digital signatures—how to sign with RSA and Rabin. In *Advances in Cryptology — EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pp. 399–416. Springer, 1996. doi: 10.1007/3-540-68339-9_34. URL https://www.cs.ucdavis.edu/~rogaway/papers/exact.html.

Yves Bertot, Olivier Pons, and Loïc Pottier. Dependency graphs for interactive theorem provers. Technical Report 4052, INRIA, 2000. URL https://cedric.cnam.fr/~pons/PAPERS/RR-4052.pdf.

Sujoy Bhore, Timothy M. Chan, Zhengcheng Huang, Shakhar Smorodinsky, and Csaba D. Tóth. Sparse bounded hop-spanners for geometric intersection graphs. In *41st International Symposium on Computational Geometry (SoCG 2025)*, volume 332 of *Leibniz International Proceedings in Informatics*, pp. 17:1–17:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi: 10.4230/LIPIcs.SoCG.2025.17.

Arthur Charguéraud and François Pottier. Verifying the correctness and amortized complexity of a Union-Find implementation in separation logic with time credits. *Journal of Automated Reasoning*, 62(3):331–365, 2019. doi: 10.1007/s10817-017-9431-7. URL https://www.charglueraud.org/research/2017/credits_jar/credits_jar.pdf.

Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. *arXiv preprint arXiv:2203.00671*, 2022. doi: 10.48550/arXiv.2203.00671. URL https://arxiv.org/abs/2203.00671.

Rayan Chikhi and Guillaume Rizk. Space-efficient and exact de Bruijn graph representation based on a Bloom filter. *Algorithms for Molecular Biology*, 8:22, 2013. doi: 10.1186/1748-7188-8-22. URL https://pmc.ncbi.nlm.nih.gov/articles/PMC3848682/.

Andrew M. Childs, Dmitri Maslov, Yunseong Nam, Neil J. Ross, and Yuan Su. Toward the first quantum simulation with quantum speedup. *Proceedings of the National Academy of Sciences*, 115 (38):9456–9461, 2018. doi: 10.1073/pnas.1801723115. URL https://arxiv.org/abs/1711.10980.

Graham Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005. doi: 10.1016/j.jalgor.2003.12.001. URL https://www.cs.ox.ac.uk/people/graham.cormode/pubs/html/CormodeMuthukrishnan04CMJalg.html.

Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, pp. 625–635. Springer, 2021. doi: 10.1007/978-3-030-79876-5_37. URL https://lean-lang.org/papers/lean4.pdf.

Petros Drineas, Michael W. Mahoney, S. Muthukrishnan, and Tamás Sarlós. Faster least squares approximation. *Numerische Mathematik*, 117(2):219–249, 2011. doi: 10.1007/s00211-010-0331-6.

Gintare Karolina Dziugaite and Daniel M. Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2017. URL https://auai.org/uai2017/proceedings/papers/173.pdf.

Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972. doi: 10.1145/321694.321699. URL https://dl.acm.org/doi/10.1145/321694.321699.

Martin Fürer. Faster integer multiplication. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pp. 57–66. ACM, 2007. doi: 10.1145/1250790.1250800. URL https://doi.org/10.1145/1250790.1250800.

Craig Gidney and Martin Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, 2021. doi: 10.22331/q-2021-04-15-433. URL https://doi.org/10.22331/q-2021-04-15-433.

Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988. doi: 10.1145/48014.61051. URL https://doi.org/10.1145/48014.61051.

David Harvey and Joris van der Hoeven. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics*, 193(2):563–617, 2021. doi: 10.4007/annals.2021.193.2.4. URL https://annals.math.princeton.edu/2021/193-2/p04.

Jan Hoffmann and Martin Hofmann. Amortized resource analysis with polynomial potential: A static inference of polynomial bounds for functional programs. In *Programming Languages and Systems: 19th European Symposium on Programming (ESOP)*, volume 6012 of *Lecture Notes in Computer Science*, pp. 287–306. Springer, 2010. doi: 10.1007/978-3-642-11957-6_16. URL https://doi.org/10.1007/978-3-642-11957-6_16.

Albert Q. Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Guillaume Lample, and Yuhuai Wu. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=SMa9EAovKMC.

A. Karatsuba and Yu. Ofman. Multiplication of many-digital numbers by automatic computers. *Doklady Akademii Nauk SSSR*, 145(2):293–294, 1962. URL https://www.mathnet.ru/eng/dan26729.

Ulrich Kohlenbach. *Applied Proof Theory: Proof Interpretations and Their Use in Mathematics*. Springer Monographs in Mathematics. Springer, 2008. doi: 10.1007/978-3-540-77533-1. URL https://doi.org/10.1007/978-3-540-77533-1.

Richard J. Lipton. Galactic algorithms. Gödel's Lost Letter and P=NP, October 2010. URL https://rjlipton.com/2010/10/23/galactic-algorithms/. Concept by Richard J. Lipton; terminology by Kenneth W. Regan.

Benjamin Lovitz and Angus Lowe. Nearly tight bounds for testing tree tensor network states. *arXiv preprint arXiv:2410.21417*, 2024. URL https://arxiv.org/abs/2410.21417.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011. doi: 10.1137/090745854.

Sadegh Mahdavi, Branislav Kisacanin, Shubham Toshniwal, Wei Du, Ivan Moshkov, George Armstrong, Renjie Liao, Christos Thrampoulidis, and Igor Gitman. Scaling generative verifiers for natural language mathematical proof verification and selection. *arXiv preprint arXiv:2511.13027*, 2025. doi: 10.48550/arXiv.2511.13027. URL https://arxiv.org/abs/2511.13027.

Ashley Montanaro and Leo Zhou. Quantum speedups in solving near-symmetric optimization problems by low-depth QAOA. *arXiv preprint arXiv:2411.04979*, 2024. URL https://arxiv.org/abs/2411.04979.

Ryan O'Donnell and John Wright. Quantum spectrum testing. *arXiv preprint arXiv:1501.05028*, 2015. URL https://arxiv.org/abs/1501.05028.

James B. Orlin. Max flows in $O(nm)$ time, or better. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*, pp. 765–774. ACM, 2013. doi: 10.1145/2488608.2488705. URL https://doi.org/10.1145/2488608.2488705.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020. doi: 10.48550/arXiv.2009.03393. URL https://arxiv.org/abs/2009.03393.

Markus Reiher, Nathan Wiebe, Krysta M. Svore, Dave Wecker, and Matthias Troyer. Elucidating reaction mechanisms on quantum computers. *Proceedings of the National Academy of Sciences*, 114(29):7555–7560, 2017. doi: 10.1073/pnas.1619152114. URL https://doi.org/10.1073/pnas.1619152114.

Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025. doi: 10.48550/arXiv.2504.21801. URL https://arxiv.org/abs/2504.21801.

Arnold Schönhage and Volker Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7 (3–4):281–292, 1971. doi: 10.1007/BF02242355. URL https://doi.org/10.1007/BF02242355.

Stephan Schulz. E—a brainiac theorem prover. *AI Communications*, 15(2–3):111–126, 2002. doi: 10.3233/EAI-2002-260. URL https://doi.org/10.3233/EAI-2002-260.

I-Hsiang Wang, Shao-Lun Huang, Kuan-Yun Lee, and Kwang-Cheng Chen. Data extraction via histogram and arithmetic mean queries: Fundamental limits and algorithms. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pp. 1386–1390, 2016. URL https://homepage.ntu.edu.tw/~ihwang/Research/Eprints/ISIT16DE.pdf.

Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7(3):36:1–36:53, 2008. doi: 10.1145/1347375.1347389. URL https://doi.org/10.1145/1347375.1347389.

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 32353–32368, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/d0c6bc641a56bebee9d985b937307367-Abstract-Conference.html.

Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 21573–21612, 2023. URL https://arxiv.org/abs/2306.15626.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. ProcessBench: Identifying process errors in mathematical reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1009–1024. Association for Computational Linguistics, July 2025. doi: 10.18653/v1/2025.acl-long.50. URL https://aclanthology.org/2025.acl-long.50/.

CONTENTS OF THE APPENDICES

A Additional related work **15**

A.1 Automated theorem proving and proof generation 15

A.2 Formal verification, proof checking, and dependency analysis 15

A.3 Automatic complexity and resource analysis 16

B Task and Dataset Construction Details Section 2 **17**

B.1 ConstEval: Benchmark Construction 17

B.2 Worked construction cases 17

B.3 ConstAble: Recovery Pipeline 18

C Extended Analysis of Recovered Constants from Section 3 **19**

C.1 Constant Distribution Analysis 19

C.2 Correlation Analysis 20

C.3 Analysis of Large Constants 21

C.4 Benefits of Constant Analysis 23

D Evaluation Details and Extended Results from Section 4 **26**

D.1 Evaluation Setup and Metrics 26

D.2 Model Comparison under the Codex Pipeline 28

D.3 Pipeline versus One-shot Inference 29

D.4 Numerical Comparison of Recovered Bounds 29

E Ablation Studies and Inference Baselines from Section 4 **31**

E.1 Baseline Protocols and Inference Budgets 31

E.2 Evaluation and Aggregation 31

E.3 Enhanced Inference Baselines 32

E.4 Pipeline Component Ablations 32

E.5 Completed One-shot Targets 34

F Constant Recovery Across External Theorems **34**

G Prompt details **35**

G.1 One-Shot Prompt 35

G.2 Expanded Baseline Prompts 36

G.3 Pipeline Prompt Protocol 36

A Additional related work

Hidden-constant recovery connects to several established approaches to mathematical reasoning and quantitative analysis. The distinction concerns the input, the requested output, and the reconstruction needed between them. Our input is a supplied proof document with a target resource model and prepared premises; our output must expose the numerical bounds and validity conditions suppressed by its asymptotic statements. The core difficulty is reconstructing these local quantitative arguments and composing them across nested proof dependencies while preserving parameter substitutions, assumptions, and the meaning of the charged resource.

A.1 Automated theorem proving and proof generation

Automated theorem proving searches for a derivation of a specified conjecture from available premises. Classical systems such as E use saturation and superposition for first-order reasoning with equality (Schulz, 2002). Language-model-based approaches guide proof search or generate proof steps in formal environments. GPT-f generates proofs in Metamath (Polu & Sutskever, 2020); LeanDojo's ReProver retrieves relevant library premises for proving Lean theorems (Yang et al., 2023); and DeepSeek-Prover-V2 combines subgoal decomposition with reinforcement learning for formal proof generation (Ren et al., 2025). These approaches address the difficulty of finding valid reasoning steps, choosing useful premises, and completing a proof under a formal specification.

For constant recovery, the supplied proof already provides an argument for the asymptotic claim, but leaves part of its quantitative content implicit. Establishing $T(x) = O(g(x))$ alone does not deliver the explicit coefficient and admissible domain required by Equation 1. A local lemma may hide a coefficient, a recursive call may change the input scale, and an absorption step may require a threshold that affects the final guarantee. Our task must reconstruct those quantities and justify their composition along the supplied dependencies. This is a difference in the requested result: theorem provers can prove explicit inequalities, and suitable procedures can construct witnesses to existential statements. They could therefore support recovery once the relevant quantitative obligations are expressed; success on a theorem-proving target alone does not establish completion of our recovery target.

Autoformalization. Autoformalization helps connect informal mathematical documents to these formal tools. Wu et al. (2022) translate natural-language mathematical statements into Isabelle/HOL specifications, while Draft, Sketch, and Prove uses informal proofs to generate formal sketches whose gaps are discharged by automated provers (Jiang et al., 2023). Such techniques could help formalize recovered derivations. Faithfully translating an asymptotic statement, however, does not itself require recovering an explicit coefficient or threshold. The missing numerical reasoning must also be reconstructed and included in the formal target and proof.

A.2 Formal verification, proof checking, and dependency analysis

Formal verification establishes that a mathematical development or program satisfies a stated specification. Proof assistants such as Lean check formal proofs through a small trusted kernel (de Moura & Ullrich, 2021), addressing whether a conclusion follows from the declared definitions and assumptions. These specifications can include quantitative properties: Charguéraud & Pottier (2019) verify both functional correctness and amortized complexity of an OCaml Union-Find implementation in Coq using separation logic with time credits. Their cost model explicitly counts source-level function calls. This illustrates how a formal cost semantics and suitable resource assertions enable certified complexity arguments, including the accounting needed for amortization.

Our central difficulty arises before and during the construction of such an explicit quantitative specification and derivation. In a research proof, a subroutine cost, parameter range, or intermediate inequality may be left implicit or distributed across several sections and cited results. Even a verified asymptotic statement need not expose the particular numerical bound requested by the recovery task. **ConstAble** reconstructs and composes that information under the supplied resource model; its current document-based outputs and model/expert checks do not provide kernel-checked certification. Formal verification could strengthen the guarantees of a subsequently formalized recovered proof.

Dependency analysis addresses the structural part of this problem. Bertot et al. (2000) study graphs of dependencies among theorems and definitions for interactive provers, and Alama et al. (2012) extract detailed dependencies from Coq and Mizar developments. Identifying which premises support a result helps organize a proof, but does not by itself determine the numerical contribution of each invocation. The same lemma can be used at different parameter values, repeated inside a loop, or invoked under different probability guarantees. Recovery must retain the substituted bound, invocation count, and applicability conditions, and determine whether costs add, multiply, or enter a recurrence. **ConstAble** uses the dependency graph to schedule local expansions and propagate both their explicit bounds and their conditions to the target.

Natural-language proof checking provides another complementary capability. For example, Mahdavi et al. (2025) study generative verification and selection of mathematical solutions. Such judgments assess arguments without providing the same guarantee as formal proof checking. In our setting, proof validity and quantitative completion also require separate assessment: a correctly justified conditional bound can remain incomplete if an invoked coefficient is unresolved. **ConstEval**'s proof and expansion checks reflect this distinction, and its paired defects test whether errors in the quantitative derivation are detected.

A.3 Automatic complexity and resource analysis

Automatic resource analysis infers bounds on resources consumed by programs or formal computational models. Albert et al. (2007) analyze Java bytecode by extracting cost relations that describe resource usage in terms of input sizes and deriving upper bounds from them. Automatic amortized resource analysis assigns potential to data structures and infers resource bounds through typing constraints; the polynomial analysis of Hoffmann & Hofmann (2010) extends this approach beyond linear bounds. Worst-case execution-time analysis further studies safe execution-time bounds for programs on specified hardware, accounting for program paths and architectural effects such as caches and pipelines (Wilhelm et al., 2008). These approaches already support concrete quantitative bounds. The distinction from our work is therefore not that previous resource analyses consider only asymptotic orders or ignore constants.

These representative methods start from an executable program, a cost semantics, or another structured model that makes the analyzed operations and transitions available. Our starting point is a mathematical exposition whose quantitative argument must be reconstructed: a recurrence may need to be recovered from prose, a local estimate expanded, or a cited theorem instantiated under the correct normalization and parameter regime. The target may concern oracle queries, circuit size, or a quantitative lower bound, in addition to running time and space. Applying a program analyzer to an implementation would therefore require a justified connection between that implementation's semantics and the paper's resource model and target statement.

The core difficulty is carrying complete quantitative information across these heterogeneous proof steps. Rounding, lower-order terms, failure probabilities, and validity thresholds can change the final coefficient or domain even when the asymptotic order remains unchanged. An explicit bound for one subroutine is useful only if its units and assumptions match the invocation and its contribution is composed correctly with the remaining proof. Automatic resource analyses could supply or certify such local bounds, while **ConstAble** organizes their use together with other mathematical dependencies. In **ConstEval**, checked external premises and stipulated costs are prepared in advance where needed; the evaluated task is the remaining quantitative reconstruction and composition.

Proof mining. Proof mining has a particularly close quantitative objective. Using proof-theoretic methods such as functional interpretations, it extracts effective bounds and other computational information from mathematical proofs that may be presented nonconstructively (Kohlenbach, 2008). The shared aim is to expose quantitative content latent in an existing argument. Proof mining develops logical extraction methods and metatheorems under specified hypotheses; our work studies an LLM-assisted workflow and benchmark for recovering explicit resource bounds from supplied TCS proof documents. **ConstAble** does not implement a general proof-theoretic extraction procedure, and its acceptance checks do not replace such guarantees. The two directions could complement each other when a local nonconstructive argument requires a quantitatively effective replacement.

B TASK AND DATASET CONSTRUCTION DETAILS SECTION 2

Section 2 defines the recovery task and its acceptance scope. The target specification fixes the requested quantity, parameter domain, resource model, and guarantee mode; these must remain consistent through source preparation, reconstruction, and evaluation. The checks below supplement that definition. Appendix D.1 gives the operational criteria for complete answers and delivered proofs.

B.1 CONSTEVAL: BENCHMARK CONSTRUCTION

Table 3: **ConstEval** composition. Each document contributes a constant-recovery instance and a supplementary negative instance with the same target.

Source category	Documents	Instances
Textbook algorithms (constructed expositions)	26	52
Research-level algorithms	23	46
Complexity theory	22	44
Quantum computation	22	44
Total	93	186

Section 2.1 specifies the paired-defect construction and totals. The 116 faulty blocks comprise numerical or algebraic faults (48); rounding, domain, or threshold faults (24); logical or dependency faults (18); algorithmic, recurrence, or probabilistic-accounting faults (16); and external-result or evidence-completeness faults (10). These block counts are distinct from the 147 atomic error annotations. Each negative document retains the target and overall proof route while changing one or two localized argument blocks. Its negative label concerns the derivation, even if the theorem is true.

Two independently diagnosable defects: Dinic. When two blocks are modified, each must be independently diagnosable and sufficient to undermine the target argument. In the Dinic instance, one block restricts a running-time proposition from $m \geq 1$ to $m \geq 2$, leaving the final theorem's $m = 1$ case unsupported. The other uses $H = (n-1)m$ in a canonical cost calculation despite its premises $H \leq (n-1)S$ and $S = 2m$, dropping a factor of two from the prescribed envelope. These create separate domain-coverage and cost-accounting defects.

Evidence and annotation. Every defect must be diagnosable from the supplied document. Its annotation records the original and modified text, source anchor, atomic error, consequence for the target, and minimal repair. An annotation is labeled `FOUND` when the output identifies that error; the verification and coverage denominator are given in Appendix D.1.

B.2 WORKED CONSTRUCTION CASES

What the solver must recover. For the geometric hop-spanner target (Bhore et al., 2025), the source invokes a grouping argument for an $O(s^{3/2})$ edge count. With $s \geq 1$ vertices, a clique X, group size $b = \lceil \sqrt{s} \rceil$, and $g = \lceil |X|/b \rceil$ groups, the solver must count within-group edges, neighbor links, and a connecting star to derive

$$g\big(b^2/2 + s\big) + s \leq 6s^{3/2}.$$

It must then compose this local coefficient with the geometric hitting-point count and recurrence. In contrast, the streamed-planarity target supplies a declared Sunflower-SEFE cost of $65536p$ operations on p records; composing that contract with reduction and bookkeeping costs yields $787840(n+m)$. The resulting coefficient is conditional on that supplied cost model.

Preserving conditions across imported results. Numerical reconstruction also requires matching assumptions. In the least-squares source (Drineas et al., 2011), the main proof invokes an appendix covariance theorem and an external concentration lemma. A correct application must preserve spectral-norm normalization, the column-sampling lower bound, and failure probabilities. An event of probability at least 0.95 supports two conditional guarantees of 0.95 and 0.90, giving

$$0.95(1 - 0.05 - 0.10) = 0.8075$$

by a conditional union bound, without an independence assumption. This example illustrates why coefficients and applicability conditions must travel together through dependencies.

Screening and source accounting. Section 2.1 gives the source pool, retained corpus, and preparation workflow. Selection favors complete PDFs of 20–30 pages, with 18–19 and 31–32 pages as backup ranges; Table 3 gives the source-category composition. Page counts for screening include references and appendices. The limit of three distinct substantive external results applies across all recursively traced dependencies: repeated uses of the same external result count once. Excluded many-parameter targets include those with seven or eight input parameters. Notation such as $O^*(\cdot)$ can suppress additional parameter-dependent expressions and is therefore unsuitable for isolating the intended numerical coefficient without further specification.

Checking supplied quantitative premises. Human reviewers use GPT-5 Pro to check each supplied premise against the source proof and resource model. For coefficients already explicit in the source or imported from external results, they check the numerical justification, hypotheses, and excessive slack; for stipulated costs, they check consistency with the reference model.

Clarifications and corrections must be supported by the source argument. Any remaining specification or proof gap is resolved before accepting a positive instance; controlled defects are introduced only after this preparation. The reference-withheld reconstruction and the requirement to match growth term, cost assumptions, and guarantee mode are described in Section 2.1. The streamed-planarity, hop-spanner, and least-squares examples above distinguish stipulated subroutine costs, internally derived constants, and conditions on imported results.

Reconciling a numerical inconsistency in the source. In the collected quantum-walk source (Magniez et al., 2011), the error analysis uses $\beta_i = 18\gamma/(4\pi^3 i^2)$ for precision parameter γ, whereas the cost analysis restates it with π^2 in the denominator. The stated normalization $\frac{4}{3\gamma}\sum_{i\geq 1}\beta_i = 1/\pi$ supports the π^3 choice; the other expression gives 1. The reconstruction therefore uses the consistent schedule in both the error proof and the cost term $\log(1/\beta_i)$.

B.3 ConstAble: Recovery Pipeline

Section 2.2 and Figure 1 describe the four stages. The following details specify how their artifacts preserve the local proof and its quantitative obligations.

Target anchoring and dependency boundaries. An optional theorem identifier helps the roadmap locate the target. Graph construction starts from its most formal paper-local statement and traces the nearest invoked results backward. Definitions and setup stay in a separate registry; routine calculations, displayed estimates, and standard mathematical tools remain local to the proof. Joint-derivation nodes represent nontrivial combinations of results. An atomic node may still contain hidden constants and is marked either for quantitative resolution or for use as an explicit black box.

Local proof extraction. For each node requiring resolution, the extractor uses the article, roadmap, and local dependency graph G_i to produce a compact proof P_i. It treats direct dependencies as established black boxes and retains assumptions, parameter choices, substitutions, inequalities, case distinctions, and closure steps. Transitive proofs remain at their own nodes. If the article supplies no local derivation, extraction returns no proof.

Resolution and retained obligations. After resolving a node's direct dependencies, the resolver expands its local constants, thresholds, absorption steps, and probability bounds. It substitutes dependency parameters and combines their applicability requirements into an admissible condition. External bounds and stipulated subroutine costs enter as atomic inputs with their conditions; unsupported premises remain unresolved obligations. The expanded proof P_i^+ updates the node's core statement while preserving its source anchor, assumptions, and graph relations. The final target derivation and retained dependency proofs together provide the calculations required by Section 2.

Review and revision. Stage-specific checks examine target and proof-route coverage, graph completeness, extraction faithfulness, and quantitative consistency. Appendix E.4 reports component ablations; Appendix D.1 describes the separate final proof and expansion checks and expert adjudication.

C Extended Analysis of Recovered Constants from Section 3

Of the 91 numerical coefficients, 69 (approximately 76%) are below 1,000. The largest reaches 1.27×10^{39}. These magnitudes lead us back to the proofs: which steps create large factors, and how do explicit constants change the guarantees available at finite input sizes?

C.1 Constant Distribution Analysis

The analysis uses a task-based grouping rather than the source-category grouping in Table 3: targets [21]–[24] move from complexity theory to research-level algorithms. Excluding the two qualitative complexity targets [35] and [38] then changes the research counts from 23/22/22 to 27/16/22 (algorithms/complexity/quantum). The 93 targets provide 91 numerical coefficients: 26 textbook targets and 65 research targets, comprising 27 research-level algorithms, 16 complexity-theory results, and 22 quantum results. Targets [35] and [38] give qualitative statements without a numerical coefficient.

For a monomial, we extract its external coefficient and retain the parameter factors in the target growth term. For a polynomial, we combine like terms, remove lower-order terms, and take the largest coefficient among the retained leading terms. For a parameter-dependent expression, we retain $C(\theta)$ and evaluate its numerical value at a stated admissible reference setting. The settings accompanying Table 7 specify these choices. Table 4 pairs representative coefficients with their growth terms and charged units; square brackets identify corpus targets.

Table 4: Selected moderate recovered coefficients. Parentheses identify the charged resource under each target's cost model; nonintegers are rounded to three significant figures. D_{tree} is decision-tree depth, and $\bar{k}$ bounds the number of 1s in an N-bit string.

Algorithm	Growth term g	Constant C (charged unit)
Textbook algorithms		
Floyd–Warshall [63]	n^3	123 (steps)
String hashing [84]	n	122 (steps)
Research-level algorithms		
Matching reconfiguration [02]	$n^{9/2}$	6 (steps)
Eulerian cycle [93]	m	52 (steps)
Graph sum labels [24]	n^{10}	91.5 (word operations)
Complexity theory		
Circuit energy [19]	D_{tree}^3	14 (active gates)
Quantum computation		
State fidelity [47]	$1/\varepsilon$	151 (queries)
Finding all 1s [48]	$\sqrt{N\bar{k}}$	4.07 (queries)

Among the 65 research targets, 48 (approximately 74%) have coefficients below 1,000, as do 21 of the 26 textbook targets (approximately 81%). Eight coefficients lie in $[10^3, 10^5)$ and 14 are at least 10^5. Figure 2 displays 90 coefficients, using reciprocals and cross markers for all 14 lower bounds.

Textbook algorithms. Twenty-one of the 26 coefficients are below 1,000. Their cost models charge actions such as scanning an edge, comparing values, and updating an array. These charges give coefficients of 123 for Floyd–Warshall, 122 for string hashing, 24 for Euclid's algorithm, and 313 for suffix-automaton construction.

Research-level algorithms. Twenty of the 27 coefficients are below 1,000. The Eulerian-cycle bound is $52m + 5$ steps, including outputting every edge, and fair allocation has a coefficient of $81/16$ in its comparison bound. This group also contains geometric matching at 1.27×10^{39}, data-oblivious graph algorithms at 8.44×10^{14}, and streaming storage at 8.89×10^{13}. In [92], the coefficient 16 measures approximation quality in $w(H) \leq 16(n^{\varepsilon} - 1)\,\mathrm{OPT}_h\,/\varepsilon$.

Complexity theory. Fifteen of the 16 coefficients are below 1,000. The charged resources differ: 14 multiplies D_{tree}^3 in the circuit-energy bound, with its growth term measured in decision-tree depth. In lower bounds, small multiplier or exponent coefficients delay the scale at which the guarantee becomes strong. The Boolean-rank result [37], for example, has exponent coefficient $1/6522434150400000$ on the infinite family of ranks produced by its construction.

Quantum computation. Thirteen of the 22 coefficients are below 1,000. Fidelity estimation has query coefficient 48π, and finding all 1s has gate coefficient $178 + 35\pi$. The largest values occur in nested-update three-distinctness, pointer functions, and quantum money, where repeated updates, search, or sampling compound local costs.

All four groups contain coefficients below 1,000. The wide spread within research-level algorithms and quantum computation motivates the mechanism analysis in Appendix C.3.

C.2 Correlation Analysis

We relate all 91 coefficients to proof-graph node count, distinct directed edges, longest dependency-path length, and maximum fan-in. Paper length includes references and appendices for all 65 research targets. We use Spearman correlations, 99,999 two-sided permutations, 10,000 paired bootstrap resamples, and Holm correction across the five tests in Table 5. The graph associations are positive but modest; the paper-length estimate is inconclusive. Nodes and edges themselves correlate at 0.990, so their evidence largely overlaps.

Table 5: Associations of recovered hidden constants with paper length and proof-graph structure. N_{obs} counts targets with both a hidden constant and the indicated measure; paper length is defined only for research papers.

Measure	N_{obs}	ρ_s	95% bootstrap CI	Holm-adjusted p
Paper pages N_{page}	65	-0.103	$[-0.329,\ 0.132]$	0.413
Nodes N_{node}	91	0.247	$[0.033,\ 0.445]$	0.053
Edges N_{edge}	91	0.266	$[0.055,\ 0.460]$	0.045
Depth D_{dep}	91	0.212	$[-0.002,\ 0.418]$	0.089
Maximum fan-in	91	0.278	$[0.074,\ 0.467]$	0.042

The node correlation is 0.051 among research papers and 0.578 among textbook targets. Controlling for research-paper length gives 0.038, with a 95% bootstrap interval of $[-0.232, 0.297]$. Edge count and maximum fan-in have adjusted p-values of 0.045 and 0.042. The pooled association motivates inspection of dependency costs, while the research-only estimates limit an interpretation based on graph size alone. The remaining analysis below gives sensitivity checks and example graphs. Numerical cost traces identify the mechanisms below.

The unit of analysis is a target. The response is its original numerical coefficient in the complete 91-entry table. We count proof nodes including the target, deduplicate directed edges, and measure depth as the longest path to the target in edges. Maximum fan-in is the greatest number of direct premises of any node. A single-node graph has zero edges, depth, and fan-in. All analyzed graphs are acyclic and all counted nodes can reach the target. Paper length is total pages including references and appendices, defined for all 65 research targets.

We compute Spearman's ρ_s with average ranks for ties. A logarithm of the positive coefficient leaves this rank statistic unchanged. For each test, 99,999 random permutations yield a two-sided p-value $(b+1)/100000$, where b counts permuted absolute correlations at least as large as observed. We form marginal percentile intervals from 10,000 paired bootstrap resamples and apply Holm correction to the five tests in Table 5.

The graph examples illustrate shared premises, branching, and long chains among proof modules. Sampling, amplification, and enumeration costs within these modules are traced in the numerical expressions below.

The node correlation changes little after removing the largest coefficient ($\rho_s = 0.236$); its full-sample leave-one-target-out range is $[0.225, 0.277]$. For the 68 resource-upper-bound coefficients, it is 0.303.

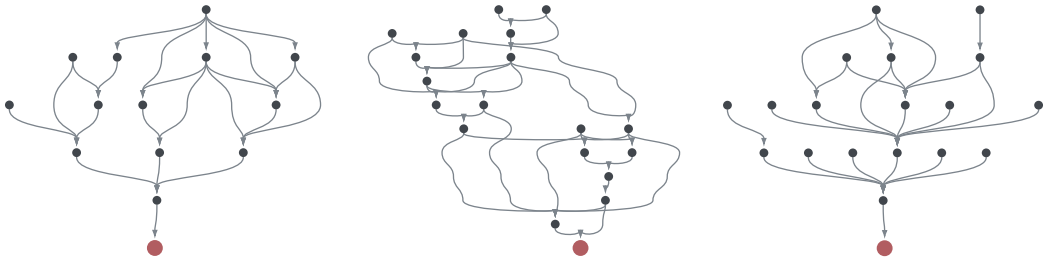


Figure 3: Complete proof-dependency graphs produced by ConstAble, illustrating the objects underlying Table 5. Gray nodes denote proof units and red nodes the target; arrows point from support to dependent. Counts refer to complete graphs; depth is the longest-path edge count.

For an area-conditioned check, we center the full-sample ranks of each graph measure and of the coefficient within the four analysis groups, then permute coefficient ranks only within groups. The residual-rank correlations are 0.263, 0.268, 0.215, and 0.278 for nodes, edges, depth, and maximum fan-in; their Holm-adjusted p-values are 0.060, 0.060, 0.063, and 0.056. The research-only page-adjusted node estimate instead uses residuals from regressing ranks on the page-count rank, with paired bootstrap resampling.

C.3 Analysis of Large Constants

Large coefficients arise when local costs multiply, when fixed auxiliary parameters require extensive enumeration, or when converting between resource models introduces additional factors. Table 6 gives representative cases. The recovered expressions identify the proof steps responsible for their magnitudes.

Table 6: Large hidden constants (charged units in parentheses), displayed to three significant figures. The rows illustrate distinct sources of large coefficients in the 91-target numerical sample.

Algorithm	Growth term g	Constant C (charged unit)
Language tester [07]	$\dfrac{m^5 2^{3m^2} (\log_2 n)^6 \log_2(1/\eta)}{\varepsilon^4}$	8.89×10^{13} (words)
Dynamic volume [10]	$\varepsilon^{-2} \log_2^2(n/\varepsilon)$	2.23×10^5 (steps/update)
Quantum money [52]	$f^{-1}\varepsilon^{-4} \ln^2(1/\eta)$	7.55×10^7 (validations)
Pointer functions [54]	$(\sqrt{n} + \sqrt{m})\Lambda^3$	6.87×10^{12} (queries)
PDQP construction [46]	N	1.70×10^7 (oracle steps)
Three-distinctness [62]	$n^{5/7}$, $k = 3$	2.54×10^7 (queries)

Here $\Lambda = \log_2(16(n+1)(m+1))$. For [46], N is the input-description size and the full bound also has an additive term. Row [62] specializes the parameter-dependent coefficient to $k = 3$, as analyzed in Appendix C.4.

Nested iteration. Geometric matching [08] has an expected-time bound $C(\omega)\rho^{3\omega/2}n^{\omega/2}$. Put $\Delta = 1 - (15/16)^{\omega/2-1}$. Its coefficient is

$$C(\omega) = 2^{24\omega}\left[3 + \frac{2^{32}5^{\omega}}{\Delta^2} + \frac{1}{\Delta}\max\left\{\frac{2^{36}5^{\omega}}{\Delta^2} + 2^{10}5^{\omega} + 2^{16}, 2^{6-11\omega/2}\right\}\right] + \frac{2(2^{24} + 2^{20})}{(\omega/2 - 1)\ln 2}.$$

At $\omega = 3$, $C \simeq 1.26943 \times 10^{39}$; the growth expression retains $\rho^{9/2} n^{3/2}$. The factors 2^{72} and Δ^{-3} expose how the prefactor and repeated inverse-gap factors compound. The expression diverges as $\omega \downarrow 2$.

In quantum money [52], the per-observable sample count has coefficient

$$A = \frac{18432(\ln 6 + 1)}{\pi^2}, \qquad C = \frac{9\pi^2}{32} A^2 \simeq 7.5456 \times 10^7.$$

The square arises because the number of samples enters both the number of validations and the accuracy allocated to each validation. The growth term is $f^{-1} \varepsilon^{-4} \ln^2(1/\eta)$, retaining failure and accuracy parameters. The streaming tester [07] has coefficient 88875640160256 in its word-space bound; the coefficient is measured against its full parameter-dependent storage expression.

For pointer functions [54], let $L = \log_2(16(n+1)(m+1))$ and $K = 25 \cdot 2^{38}$. The coefficient recorded against $(\sqrt{n} + \sqrt{m})L^3$ is $K \simeq 6.87 \times 10^{12}$. A finite-size envelope retains $(L+1)^3$, obtained from the ceiling logarithm.

Resource conversion can dominate the representative value. For data-oblivious graph algorithms [17], the selected I/O coefficient at $\epsilon = 1/2$ and minor-class sparsity parameter $d_{\mathcal{C}} = 3$ is

$$\frac{9}{2} \cdot 93724776333552 \cdot (1/2)^{-1} = 843522987001968.$$

The factor $9/2$ converts the internal expression to the sorting-cost normalization across both memory regimes. This I/O bound uses the stated density thresholds. For dynamic volume [10], update, query, and space coefficients are 223450, 1959, and 159420, respectively. The representative value is their maximum, with each resource retaining its own growth term.

Brute-force enumeration. In BINARY-CSP [23], let $M = 2^q - 1$ and $F = (d+1)^2 q^2 + d + 1$. The recovered ledger is

$$T \le 8nM + 16nq + 28n + 9 + \frac{31n^2 + 31n - 6}{2} + 66(n-1)M^{d+1}F + 11M + 6.$$

With q, d fixed, the leading coefficient relative to $M^d n^2$ is $31/(2M^d)$, giving $31/6$ at $q = 2, d = 1$. Yet the linear term contains enumeration over M^{d+1} mask tuples. Increasing q or d can make this term large while reducing the normalized quadratic coefficient.

The same distinction appears in quotable signatures [04]. With B the maximum of the supplied input and adversary cost parameters, the two recovered branches are

$$T_1 \le 3B^3 + 15B^2 + 268470713B + 671088643,$$
$$T_2 \le 3B^3 + 18B^2 + 268470741B + 134235795.$$

The representative coefficient is 3, while the lower-order terms govern substantial finite ranges.

Large quantum coefficients retain logarithmic and padding costs. The nested-update three-distinctness target [56] records

$$C = 3^{5/7} \cdot 13414602023894205333504 \simeq 2.94 \times 10^{22}$$

against $n_0^{5/7} \log_2^6 n_0$. The factor $3^{5/7}$ comes from expressing the padded length $n = 9\lceil n_0/3 \rceil$ in terms of the original length.

Other mechanisms and their interactions. The same mechanism extends to galactic algorithms, whose asymptotic guarantees can coexist with enormous finite-size costs (Lipton, 2010). Almost-linear network flow exhibits it across two hierarchies. In Chen et al. (2022, Claim 5.8 and Lemma 7.9), let P bound static embedding-path length, L count dynamic sparsification levels, and d be the tree-chain depth. Dynamic embedding gives $\Gamma = 2^L P^{L+1}$; tree-chain routing compounds this factor across d levels; the iteration bound then depends on κ^{-2}, where κ is the cycle-approximation quality. With other stretch factors kept separate, the path contribution follows

$$P \longrightarrow \Gamma = 2^L P^{L+1} \longrightarrow \Gamma^d \longrightarrow \Gamma^{2d},$$

giving degree $2d(L+1)$ in P, or P^{16} at $L=3, d=2$. Expanding Algorithm 7's potential argument also yields a conservative bound on the number J of interior-point iterations for an initialized instance with m edges and integral data bounded by U:[2]

$$J \le \left\lceil 8 \times 10^{11}\, m[\ln(mU)]^3 \kappa^{-2} \right\rceil + 1.$$

The hidden constant in this iteration bound comes from step normalization and the potential budget; κ^{-2} retains the hierarchy's input-dependent loss. The remark after Lemma 7.17 identifies a removable $\widetilde{O}(k)$ cycle-extraction loss, where $k = m^{1/d}$; removing it would eliminate a k^2 dependence from this part of the iteration bound.

Matrix multiplication magnifies the block dimension through tensor powers. The optimistic Coppersmith–Winograd example evaluated by Alman and Yu (2024, Appendix A) achieves an exponent of about 2.5 with

$$\begin{aligned} N = S = 250 \quad &\longrightarrow \quad Q = 8^{NS} \simeq 1.33 \times 10^{56443} \\ &\longrightarrow \quad Q^2 \simeq 1.77 \times 10^{112886}. \end{aligned}$$

Here Q is the block-identity dimension, and $O(Q^2)$ is the prefactor dependence of the standard recursive conversion.

Integer multiplication connects a contraction requirement to an exponential validity threshold. In the basic construction of Harvey and van der Hoeven (2021, Section 5), transform, packing, precision, and length factors combine as $K = 12 \cdot 3 \cdot 6 \cdot 8 = 1728$. The normalized cost $F(n) = M(n)/(n \log n)$ contracts when $K/(d - 1/2) < 1$, leading to

$$K = 1728 \longrightarrow d = 1729 \longrightarrow n_0 = 2^{d^{12}}, \qquad \log_{10} n_0 \simeq 2.15 \times 10^{38}.$$

The threshold n_0 enters the uniform hidden constant $C = \max\{B, 5000A\}$ through $B = \max_{2 \le n < n_0} F(n)$, while A bounds the additive recurrence term. The same paper outlines an optimized construction with $d = 9$.

C.4 Benefits of Constant Analysis

Explicit constants let us compare guarantees at a given input size, tighten proof steps, and translate accuracy or success requirements into resource budgets.

Finite-Size Comparisons For power-law bounds $C_1 n^a$ and $C_2 n^b$ with $a < b$, the intersection is $n_* = (C_1/C_2)^{1/(b-a)}$. Figure 4 compares Dijkstra implementations and quantum versus classical queries. The left column sets constants to one; the right restores the recovered coefficients and lower-order terms.

Dijkstra. The selected expressions are

$$U_{\text{sparse}} = 1201m + 1794n + 968n \log_2 n, \qquad U_{\text{dense}} = 152(n^2 + m).$$

At $n = 4096$, these become $1201m + 54927360$ and $152m + 2550136832$. Their intersection is $m_* = 2495209472/1049 \simeq 2378655$, about 14.18% of the $n(n-1)$ possible edges in a simple directed graph. Above this density the dense upper bound is smaller.

Finding all 1s. For $N = 2^r$, $r \ge 1$, and an upper bound $\bar{k} \ge 1$ on the number of ones, the exact-success quantum query bound is $((\pi + 5)/2)\sqrt{N\bar{k}}$. At $\bar{k} = 16$, it intersects the exact classical worst-case lower bound N at $N_* = 16((\pi + 5)/2)^2 \simeq 265.14$. The first allowed power of two above this value is 512.

[2]The calculation assumes the paper's initialization and stability guarantees and successful oracle calls. For $mU \ge 2$ and $0 < \kappa \le 1$, Algorithm 7 uses $\alpha = 1/[1000 \ln(mU)]$ and update normalization $-g_{\text{alg}}^\top h = (\alpha\kappa)^2/800$, where g_{alg} is the maintained gradient. Write $\beta = \alpha\kappa$ and let $\mathbf{L}_{\text{true}}$ be the diagonal matrix of current edge lengths. Lemma 9.1 and Equation (45) give $\|\mathbf{L}_{\text{true}} h\|_1 \le \beta/50$ and a true-gradient contribution at most $-21\beta^2/40000$. The barrier remainder from Lemma 4.5 is at most $\beta^2/125000$, while the logarithmic term is concave. Each step therefore decreases the potential by at least $\beta^2/2000$. A decrease exceeding $400m \ln(mU)$ suffices to reach the target residual $(mU)^{-10}$.

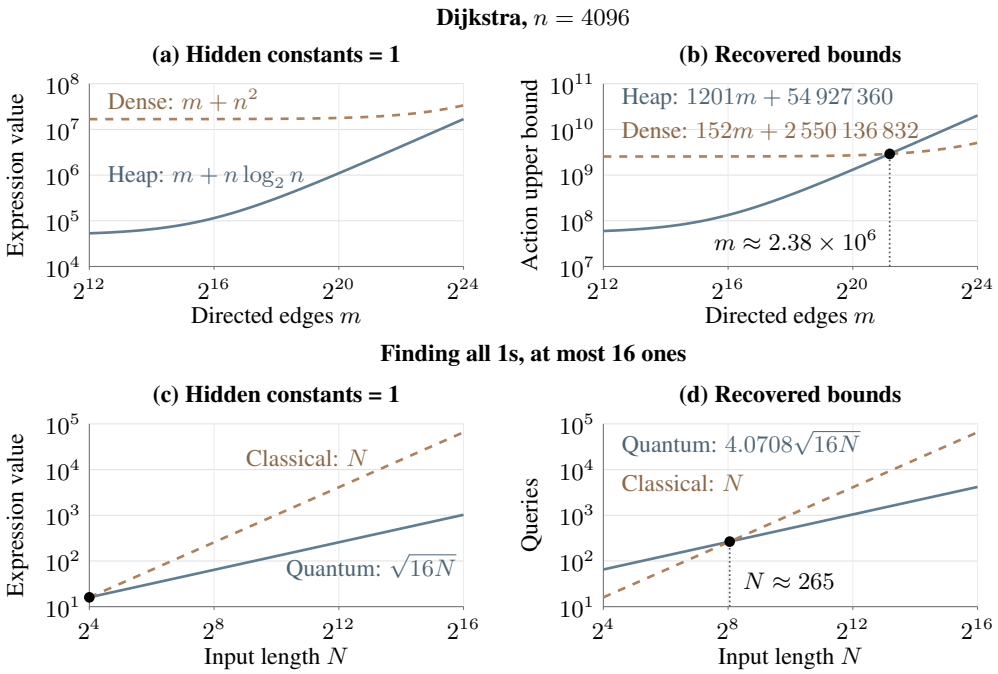


Figure 4: Finite-size bound comparisons with constants set to one (left) and recovered coefficients (right). Top: Dijkstra upper bounds intersect near $m = 2378655$ at $n = 4096$. Bottom: the quantum upper bound crosses the exact classical lower bound near $N = 265$, versus $N = 16$ with unit constants; the first certified power of two is 512. Axes are logarithmic; quantum curves interpolate between powers of two.

Proof Refinement Tracking the growth term through each input-size regime prevents unnecessary inflation of a uniform coefficient. Two derivations for target [62] must be distinguished. Post hoc refinement of an earlier recovered bound. The calculation in Section 3.3 starts from the earlier common coefficient $B = 2^{68} \simeq 2.95 \times 10^{20}$. Its supplied multiplicity estimates and learning-graph conversion give $50331648 \cdot n^{5/7}$ queries above B, with a linear scan below B, for success probability at least $3/4$. The author analysis retains these premises and normalizes the scan: $n/n^{5/7} = n^{2/7} < B^{2/7} \simeq 705639 < 50331648$. Thus 50331648 covers both regimes. This tightens that earlier guarantee; it is not a new model output or an improvement over the current inventory entry. The sharper current inventory bound. For prior-knowledge k-distinctness [62], write $\alpha_k = 1 - 2^{k-2}/(2^k - 1)$. The recovered query bound uses

$$C(k) = \max\left\{2^{20}k^2\left(1 + \frac{k(k-1)}{2}\right)^{\alpha_k},\ 2^{10\cdot 2^{k-2}}\right\}, \quad Q \le C(k)n^{\alpha_k},$$

with the supplied prior estimates and success probability at least $3/4$. Thus $C(3) = 2^{20} \cdot 9 \cdot 4^{5/7} \simeq 25403015.53$, while $C(4) = 2^{40}$. The second term absorbs the linear scan in the small-input regime: dividing its cost n by n^{α_k} leaves $n^{1-\alpha_k}$. At $k = 3$, the first term determines the common coefficient.

Quantum-walk search [57] has coefficient $C_0 = 1 + 27\pi^2(3 + \log_2 \pi) \simeq 1240.53$ for single-run success at least $1/24$. Repeating independently until failure is at most $1/3$ requires $\lceil \ln 3/\ln(24/23) \rceil$ runs. The ceiling relaxation gives the sufficient coefficient $C_0(1 + \ln 3/\ln(24/23)) \simeq 33262.88$. Explicit success levels therefore belong beside the cost expression when comparing bounds.

Resource Planning An explicit bound converts a target accuracy or success probability into a sufficient resource budget. The following applications illustrate the quantities such a calculation needs.

Quantum computation and scientific simulation. Query bounds are one layer of this accounting. Gidney & Ekerå (2021) translate logical-operation counts for factoring into qubit and runtime

estimates under stated hardware assumptions, including error correction and repeated attempts. In simulation, constants in approximation bounds determine how many time slices or circuit segments a chosen accuracy requires. Childs et al. (2018) compare explicit gate counts for three Heisenberg-chain simulation methods at error 10^{-3}. For FeMoco, Reiher et al. (2017) allocate error among phase estimation, time-evolution approximation, and gate synthesis; their 54-orbital serial calculation at 0.1 millihartree precision yields an extrapolated estimate of 1.1×10^{15} logical T gates. These calculations connect proof-level accuracy requirements to a concrete computation budget.

Security and statistical guarantees. Success and confidence requirements impose analogous budgets. When additive loss is negligible, a security reduction with multiplicative loss 2^{20} requires the primitive's attack-advantage bound to be 2^{20} times smaller than the desired protocol-level bound. For RSA and Rabin signatures, Bellare & Rogaway (1996) show how explicit reduction losses relate signature security to the underlying problem. In learning, given an error bound $C/\sqrt{n}$, choosing $n \geq (C/\varepsilon)^2$ is sufficient to certify error at most ε; Dziugaite & Roy (2017) demonstrate the value of numerical evaluation by optimizing PAC-Bayes bounds for neural networks. In both settings, evaluating the numerical terms determines the guarantee available at a fixed budget.

Streaming and memory budgets. For data structures, small per-item costs accumulate over the input. The Count-Min sketch uses $w = \lceil e/\varepsilon \rceil$ columns and $d = \lceil \ln(1/\delta) \rceil$ rows, storing wd counters and updating one counter in each row per item (Cormode & Muthukrishnan, 2005). In genome assembly, Chikhi & Rizk (2013) obtain expected de Bruijn graph storage of approximately $1.44 \log_2(16k/2.08) + 2.08$ bits per node, or 13.2 bits at $k = 27$. Their reported 5.7 GB peak memory for a human-genome assembly also includes other structures. Such accounting connects per-item constants to total storage while keeping analytical hidden constants and measured system costs distinct.

Numerical inventory and parameter settings. Table 7 records the complete numerical inventory. All 91 numerical entries enter the summary statistics and correlation analysis.

Table 7: Recovered coefficients for all targets. Values are rounded; dashes denote qualitative results.

ID	Coefficient	ID	Coefficient	ID	Coefficient	ID	Coefficient
01	224.7	25	0.2642	49	3	73	1148
02	6	26	1	50	49.65	74	24
03	2	27	0.03125	51	46.06	75	1020
04	3	28	4	52	7.546×10^{7}	76	601.5
05	32	29	0.02297	53	0.08333	77	455
06	7.878×10^{5}	30	1	54	6.872×10^{12}	78	670
07	8.888×10^{13}	31	4.481	55	3.602×10^{5}	79	27
08	1.269×10^{39}	32	1.085×10^{-4}	56	2.94×10^{22}	80	334
09	1.538×10^{4}	33	6.291×10^{6}	57	1241	81	166
10	2.234×10^{5}	34	47	58	0.6667	82	113
11	3	35	—	59	9.156×10^{4}	83	551
12	5.062	36	1.111×10^{-5}	60	1	84	122
13	6	37	1.533×10^{-16}	61	1.323×10^{5}	85	203
14	4	38	—	62	2.54×10^{7}	86	313
15	1.049×10^{6}	39	1.086×10^{-5}	63	123	87	76
16	480	40	7.44×10^{-4}	64	29	88	162
17	8.435×10^{14}	41	38.01	65	176	89	8
18	302	42	1	66	150	90	128
19	14	43	0.008255	67	170	91	128
20	30	44	0.0286	68	1201	92	16
21	96	45	0.01804	69	152	93	52
22	62	46	1.701×10^{7}	70	1146		
23	5.167	47	150.8	71	230		
24	91.5	48	288	72	1675		

Parameter settings for numerical entries. Evaluating $C(\theta)$ at fixed admissible parameters gives a concrete numerical reference for each bound. Structural choices specify the algorithmic regime:

$\omega = 3$ uses cubic matrix-multiplication scaling, $d = 2$ gives a planar setting, $q = 2$ gives a binary domain, and $k = 3$ specializes k-distinctness to triples. Error or failure probability $1/3$ corresponds to success at least $2/3$, providing a constant-reliability reference point for evaluating amplification costs. The parameter-dependent coefficients use the following settings: [08], $\omega = 3$; [09], dimension $d = 2$ and fatness $F = 1$; [17], $\epsilon = 1/2, d_{\mathcal{C}} = 3$; [21], $k = 3$, a single-vertex single-edge hypergraph, coincident embeddings, and certificate parameters $K_A = 8, c = 1$; [22], fixed target graph H and width $w = 2$, normalized by Ln^3 with expression length L retained; [23], $q = 2, d = 1$; and [33], an assumed input algorithm with coefficient $C = 1$ and exponent saving $\eta = 1$.

For quantum targets, the settings are [41], $\beta = 1/2, s = 1$, retaining the finite separation certificate; [50], sparsity parameter $c = 1$; [51], error $\varepsilon = 1/3$; [55], failure parameter $\delta = 1/3$; [61], $d = 2, \delta = 1/3$; and [62], $k = 3$. The MAX-k-SAT target [90] uses $k = 3, \varepsilon = 1/2, \beta = 2$, with $B = 2 - \log_2(6)/96$ and growth expression $(n^5 + mn^4)B^n$; its retained coefficients are 64 and 128. Lower-bound examples [43] and [44] use error $1/3$.

Target [89] counts edge reversals per external update, with full bound $8\lceil \log_2 n \rceil^2 - 6\lceil \log_2 n \rceil + 1$ for $n \geq 2$; [91] counts core instructions after supplied preprocessing, excluding prime finding; and [93] counts steps including output, with full bound $52m + 5$ for $m \geq 1$.

Domains of lower bounds. For [29], the recovered inequality is $B > cn^2/k^2 - 1$, with $c = (1 - 2/\sqrt{10})/16$. In [58], $2/3$ is an asymptotic leading rate, with strict form $2/3 - o(1)$. In [37], the exponent coefficient $1/6522434150400000$ holds on the infinite family of ranks specified by the construction.

D Evaluation Details and Extended Results from Section 4

This appendix specifies the evaluation criteria and numerical scoring behind Section 4, provides complete model profiles, and records additional proof cases. Inference protocols, token measurements, and component ablations are collected in Appendix E.

D.1 Evaluation Setup and Metrics

Instances and execution modes. Positive instances are documents 1–93; their paired negative instances are numbered 101–193 and retain the same target specifications. We assess bound completion on positives and defect detection on negatives. Codex runs use an agent environment with file and tool access; API runs invoke the model through the pipeline implementation. The two modes use their respective context presentation, tool interfaces, and inference settings. DeepSeek's API and one-shot experiments use version `0731`.

The six full-corpus model configurations and their reasoning settings are listed in Appendix D.2.

Fully computed bounds. A record is fully computed when every requested target clause and parameter regime has an explicit final bound supported by a complete proof. Every step of the proof must be correct, checkable, and quantitatively explicit, with no undefined variables, uncomputed constants, or unexpanded asymptotic terms. When invoking a lemma from the source paper, the output must also provide the calculations that derive that lemma's constants. Declared input parameters and auxiliary quantities with explicit definitions are allowed. An external coefficient that has only been named, without a numerical value or an explicit expression in the permitted inputs, remains unresolved.

Every output receives separate proof and quantitative-expansion checks by GPT-6 Sol. Each pass uses a separate agent instance, also distinct from those used for source preparation and answer generation. The proof check examines completeness, correctness, verifiability, and dependency premises. The expansion check covers all target clauses and parameter regimes, coefficients and thresholds, constants in invoked source lemmas, and remaining asymptotic terms.

Expert sampling, the adjudication of model judgments, and the separation of checking agents from answer-generation agents follow Section 4.1. For every article, experts randomly select one or two short GPT-5.6 Sol proof fragments for verification. Submitted answers remain fixed during adjudication.

The bound completion rate is

$$R_{\text{comp}} = \frac{1}{93} \sum_{i=1}^{93} c_i,$$

where $c_i = 1$ when the output meets all bound and proof requirements after adjudication, and $c_i = 0$ otherwise, including unfinished runs. Tables and figures report percentages.

Errors in delivered proofs. Proof-audit rates for Qwen3.7 Plus and DeepSeek v4 Flash use all 67 research-paper positive instances in each method and execution mode. The article and target files are identical across the eight settings. Each judgment covers one article–configuration output.

Operational definition of a substantive proof error. An output has a *substantive proof error* when it asserts that a conclusion has been established, but an essential step of its delivered argument is false or unsupported under its stated assumptions. A step is essential when removing it leaves that argument without support for the asserted conclusion. The claim includes the algorithm's correctness, parameter range, resource model, quantitative dependence, and success guarantee. An error finding must identify the asserted claim and the specific offending step, together with one of the following witnesses:

1. a counterexample or algebraic contradiction to a required inference, such as an invalid independence assertion or algorithmic transition;
2. an unmet premise of an invoked lemma, or a missing indispensable argument such as a state invariant, probability bound, or lifting proof;
3. an invalid change to the claimed parameter dependence or resource accounting, such as absorbing a variable factor into an absolute constant or treating a query bound as a total-running-time bound.

In this audit, a step can be supported by an explicitly cited source lemma with matching hypotheses or an identified final dependency proof from the same run. Unsubmitted reasoning and review drafts cannot fill a gap. We judge the delivered claim on its stated scope: a defective argument can support an error finding even if the theorem is true, while a valid conditional or weaker conclusion can remain valid without completing the target.

Other outcomes and counting. A *local numerical error* is an incorrect final coefficient, rounding budget, or operation count whose correction preserves the justified algorithm, parameter exponents, and success guarantee. A harmless typographical slip that leaves the final claim supported is not counted. *No proof delivered* denotes an empty output or an explicit deferral of the proof; asserting that a proof is complete while omitting its indispensable argument instead constitutes a substantive gap. If a delivered proof has both substantive and numerical errors, it is counted once in the substantive category.

Let N_{audit} denote the fixed number of research-paper tasks and N_{err} the recorded substantive errors. The error rate is

$$R_{\text{err}} = \frac{N_{\text{err}}}{N_{\text{audit}}}.$$

Here $N_{\text{audit}} = 67$ for one setting and $N_{\text{audit}} = 268$ for four pooled settings. Target completion is recorded separately by R_{comp}.

Responses to injected defects. The negative instances contain 147 atomic error annotations within 116 fault blocks. We report the *annotation coverage rate*,

$$R_{\text{cov}} = \frac{N_{\text{found}}}{N_{\text{ann}}}, \qquad N_{\text{ann}} = 147,$$

where N_{found} counts annotations labeled `FOUND`. Each annotated error is counted once when the output identifies it.

The injected defects and their annotations are fixed before evaluation. GPT-6 Sol verifies each `FOUND` label against the preset annotation and the supplied proof. TCS experts randomly audited 10% of GPT-6 Sol's verification decisions and confirmed every sampled decision.

D.2 Model Comparison under the Codex Pipeline

The six full-corpus Codex configurations are GPT-5.6 Sol and Luna (xhigh), Terra (high), Qwen3.8 27B (local, with an extended runtime allowance), DeepSeek v4 Flash (xhigh), and Qwen3.7 Plus (xhigh).

Sol leads Codex completion at 94.6%, followed by local Qwen3.8 27B at 90.3%, Luna and Terra at 88.2%, DeepSeek at 83.9%, and Qwen3.7 Plus at 58.1%. Defect coverage instead reaches 97.3% for local Qwen, 88.4% for DeepSeek, 85.7% for Qwen3.7 Plus, 81.0% for Luna, and 78.9% for Terra; Sol coverage is unreported. Figure 5 shows these profiles; The following paragraphs compare execution modes.

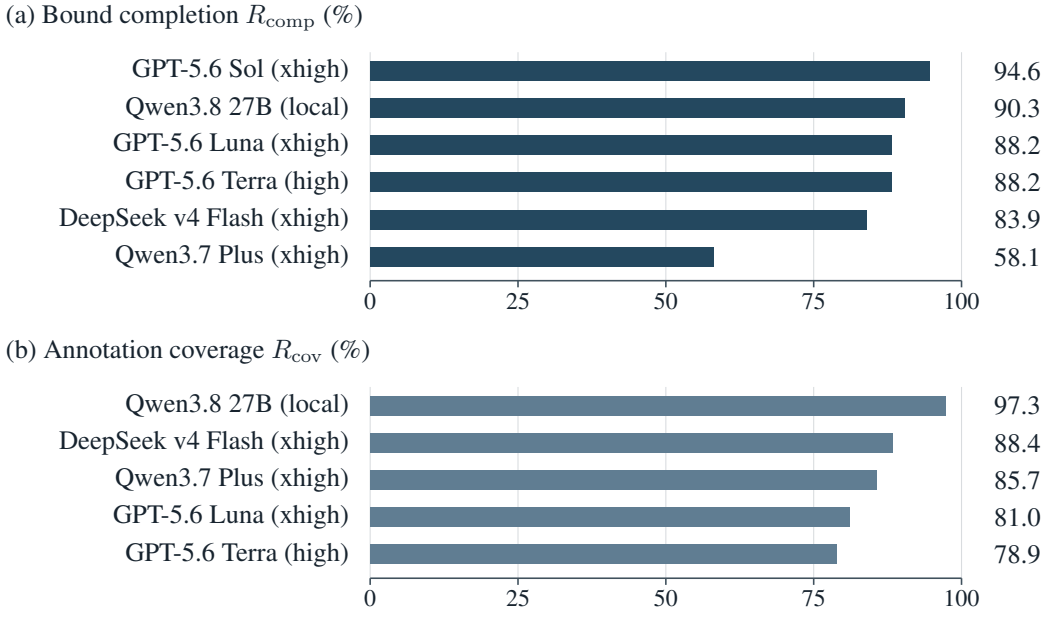


Figure 5: Codex pipeline profiles, sorted within each panel. (a) Bound completion R_{comp} on 93 positive instances. (b) FOUND coverage R_{cov} over 147 negative error annotations; Sol is unreported. Parentheses show reasoning effort or local deployment (Appendix D.1).

Tightness compares accepted completed bounds only when resource, unit, and admissible regime agree. Each target has twelve inputs: two small, two medium, and eight large. A tighter bound earns one point, a tie one half, and a loss zero. The score S_m averages successively over inputs, shared articles, and opponents, then multiplies by 100. The pool contains twelve configurations; Appendix Table 8 displays the six full-corpus Codex configurations while retaining all eligible opponents. Missing comparisons are excluded. Ties occupy 43.59%–54.87% of comparisons, and Sol and Terra differ by only 0.16 score points. The final score uses different weighting from pooled win/tie/loss percentages; Appendix D.4 specifies both and the tie tolerance. Figure 5 gives the complete completion and annotation-coverage profiles.

API and Codex comparisons. The full-pipeline API completion rates are 92.5% for DeepSeek and 65.6% for Qwen3.7 Plus, exceeding their Codex rates of 83.9% and 58.1% by 8.6 and 7.5 percentage points. Negative-annotation coverage has different directions: Codex is higher by 5.4 points for DeepSeek, while API is higher by 2.7 points for Qwen3.7 Plus (Table 1). Both mode differences are smaller than the corresponding pipeline gains over one-shot inference.

Pooling both workflows and models within each execution mode gives substantive-error rates of 7.1% (19/268) for Codex and 5.2% (14/268) for API.

Case study: an unresolved dependency (Article 19). For Boolean-function energy complexity, the DeepSeek Codex run reports $E_{\text{circ}}(f) \leq 14 D_{\text{tree}}(f)^3$, whereas Qwen Codex leaves a connector gate-count coefficient unresolved; E_{circ} denotes circuit energy and D_{tree} decision-tree depth. Under API, Qwen recovers the same coefficient 14, while DeepSeek leaves that quantity unresolved. The final

proofs tie this reversal to the numerical information retained in their dependency records: completion depends on recovering the connector coefficient and carrying it into the target bound.

D.3 Pipeline versus One-shot Inference

Table 1 reports completion, defect coverage, and the proof audit on research articles. The workflow-level comparisons are discussed in Section 4.2. Appendix E.3 gives expanded inference results, including repeated-detection scores at $h = 3$ and $h = 7$; Table 12 lists one-shot target IDs. The cases below show the proof obligations behind these aggregate outcomes.

Case study: preserving parameter dependence (Article 23). In the component twin-width problem, both Codex one-shot runs derive a cost containing $nM^{d+1}((d+1)^2q^2 + d + 1)$, where $M = 2^q - 1$, but then claim the specific $O(M^d n^2)$ target. The ratio is

$$\frac{nM^{d+1}((d+1)^2q^2 + d + 1)}{M^d n^2} = \frac{M((d+1)^2q^2 + d + 1)}{n},$$

which need not stay bounded as the width d grows. Calling the remaining factor a parameter-dependent constant does not justify the stated uniform bound. The DeepSeek Codex pipeline retains the parameter dependence in its explicit cost and leaves the stronger target unresolved.

Case study: checking a theorem's premises (Article 56). For 3-Distinctness, the Qwen one-shot proofs substitute the nested quantum-walk cost $\widetilde{O}(\sqrt{nm/s_1})$ without establishing the accepted-partition condition $n_2 \geq n/50$. That condition controls the inner marked fraction and is needed before the cost theorem applies. The Codex pipeline identifies the missing expanded setup and nested-cost dependencies and makes its numerical conclusion conditional on them.

Errors shared across workflows. In Article 15, seven of the eight settings turn an unconditional per-item failure probability into a bound on value-weighted expected loss. Rare failures can carry most of the value, so a conditional or value-weighted argument is required. The Qwen pipeline proofs in Article 22 also omit a necessary inclusion condition in a dynamic-programming transition.

D.4 Numerical Comparison of Recovered Bounds

Table 8: Numerical-bound comparisons for the six Codex configurations evaluated over the full positive corpus. Each row includes all eligible opponents from the twelve-configuration pool, including API configurations and Codex configurations evaluated on subsets. Valid comparisons $N_{\mathrm{cmp},m}$ are the denominator for win/tie/loss percentages. Rows descend by S_m (out of 100), with separate averaging over articles and opponents (Appendix D.4). Comparable subsets differ; percentages are rounded independently.

Model (setting)	$N_{\mathrm{cmp},m}$	$R_{\mathrm{win},m}$ (%)	$R_{\mathrm{tie},m}$ (%)	$R_{\mathrm{loss},m}$ (%)	S_m
GPT-5.6 Sol (xhigh)	7,191	28.59	50.30	21.11	51.67
GPT-5.6 Terra (high)	5,217	29.10	43.59	27.31	51.51
Qwen3.8 27B (local)	7,227	25.97	49.49	24.53	50.20
DeepSeek v4 Flash (xhigh)	6,951	25.41	49.03	25.56	50.17
Qwen3.7 Plus (xhigh)	5,139	22.12	54.87	23.00	48.18
GPT-5.6 Luna (xhigh)	6,807	20.49	51.67	27.84	46.47

Configuration pool and table scope. The numerical comparison uses twelve full-pipeline configurations. These comprise the six Codex configurations in Appendix D.2, three additional Codex configurations (GPT-5.5, Qwen3.8 Max, and Qwen3.8 Flash), and three API configurations (Qwen3.8 27B, DeepSeek v4 Flash, and Qwen3.7 Plus). Qwen3.8 Max and Flash were evaluated on randomly selected subsets of 11 and 37 positive instances, respectively. GPT-5.5 was evaluated on 37 selected positive instances, and Qwen3.8 27B through API on 31 selected positive instances.

Table 8 focuses on the same six Codex configurations evaluated over the full positive corpus. API configurations and the three additional Codex configurations evaluated on subsets are included as comparison opponents but are not displayed as rows. Their accepted bounds contribute comparisons

on shared instances; the comparison counts, win/tie/loss rates, and scores of every displayed row retain all eligible opponents in the twelve-configuration pool.

Input grid. We compare the numerical values of accepted bounds on a fixed input grid. Each of the 93 positive instances has twelve parameter settings: two small, two medium, and eight large, giving 1,116 article–input pairs. Scale is defined within each article's grid. Every tuple has at least one nonzero numerical parameter and retains its applicability conditions. All models use the same grid for an article; values are computed from symbolic bounds.

Pairwise scoring. We compare two configurations when both have completed bounds accepted under Appendix D.1's protocol for the same article, input, resource metric, unit, and admissible regime. The catalogue selects one resource metric per article. A smaller upper bound or larger lower bound wins one point; a tie receives half a point, and a loss receives zero. We average first over comparable inputs within an article, then over shared articles, and finally over opponents, giving equal weight at each level. Multiplying by 100 gives the score. Missing values are excluded from the corresponding averages, and an article or opponent with no comparable values is omitted. Completion is scored separately. When all twelve inputs are comparable, the small, medium, and large groups contribute $2/12$, $2/12$, and $8/12$ of that article's score.

Ties and numerical precision. For positive values x, y, the implementation declares a tie when $|\log_{10} x - \log_{10} y| \le 10^{-10}$. This tolerance avoids ranking tiny rounding differences in evaluated expressions. Large values can be compared through their stored base-ten logarithms. Nonpositive values are ordered directly by value and sign. The same comparison rule is used for every configuration; a lower-bound metric reverses the preferred ordering.

Equivalently, let $\mathcal{O}_m$ be the comparable opponents of configuration m, $\mathcal{A}_{mk}$ their shared articles with accepted completed bounds, and $\mathcal{S}_{amk}$ the comparable inputs for article a at its designated resource metric. For pairwise outcomes $w_{amks} \in \{0, \frac{1}{2}, 1\}$, the score S_m is

$$S_m = \frac{100}{|\mathcal{O}_m|} \sum_{k \in \mathcal{O}_m} \frac{1}{|\mathcal{A}_{mk}|} \sum_{a \in \mathcal{A}_{mk}} \frac{1}{|\mathcal{S}_{amk}|} \sum_{s \in \mathcal{S}_{amk}} w_{amks}. \tag{2}$$

A score of 50 represents a balanced average of wins and losses against comparable opponents, with half-credit for ties. The score records whether a bound is tighter without weighting the size of the improvement.

Comparison counts and pooled outcomes. In Table 8, $N_{\mathrm{cmp},m}$ counts valid article–input–opponent comparisons for configuration m. Let $N_{z,m}$ count outcomes of type $z \in \{\mathrm{win}, \mathrm{tie}, \mathrm{loss}\}$; their pooled rates are $R_{z,m} = N_{z,m}/N_{\mathrm{cmp},m}$. The same article can contribute multiple inputs and opponents, so larger comparable sets receive more weight in the pooled rates. The equal weighting by article and opponent in S_m can therefore give a different result from the pooled win percentage plus half the tie percentage.

For example, Sol's displayed win and tie percentages in Table 8 give $28.59 + 50.30/2 = 53.74$, whereas its article- and opponent-averaged score S_{Sol} is 51.67. Each score uses that configuration's comparable article and opponent sets. This difference follows from the weighting above, without changing any pairwise outcomes.

Case study: a tighter coefficient at the same order (Article 2). For connectivity of the odd-matching flip graph, the Sol and Luna Codex runs use the same benchmark costs and begin with the same operation count,

$$T(n) \le 3n^2 + 3n^{5/2}\big(1 + 2n(n-1)\big).$$

Luna bounds the two positive contributions separately and obtains $T(n) \le 9n^{9/2}$. Sol retains the negative term in the expanded expression and proves $-6n^{7/2} + 3n^{5/2} + 3n^2 \le 0$ for $n \ge 1$, obtaining $T(n) \le 6n^{9/2}$. The latter bound is one third smaller throughout the shared domain, so Sol wins every selected input for this metric. The gain comes from a tighter algebraic estimate within the same algorithm and cost model; each input contributes one win to the pooled count and pairwise score.

E Ablation Studies and Inference Baselines from Section 4

We compare enhanced inference baselines through API using DeepSeek v4 Flash (xhigh) and Qwen3.7 Plus (xhigh), and ablate pipeline components using Qwen3.7 Plus (xhigh). The protocols and budgets below specify how additional calls are allocated and how their outputs are scored.

E.1 Baseline Protocols and Inference Budgets

Budget calibration. The call counts are informed by API measurements pooling positive instances 1–93 and negative instances 101–193, matched to the same model's completed one-shot runs. For each run, uncached-token use is

$$B = N_{\text{tok}} - N_{\text{cache}},$$

where N_{tok} and N_{cache} are the provider-reported total tokens and cached input tokens, respectively, summed across all N_{call} model calls. The median of the instance-level pipeline-to-one-shot ratios over the pooled instances is $12.17\times$, with an interquartile range of $8.01\times$–$16.65\times$. The model-specific medians are $11.12\times$ for DeepSeek v4 Flash and $12.69\times$ for Qwen3.7 Plus. These measurements motivate fixed call counts of 12–13 for the expanded baselines; ordinary one-shot remains a single-call reference at its native token consumption.

Parallel sampling: 13 independent generations. The model independently generates 13 complete solutions from the original input. Each call receives the same task and has no access to the other outputs.

Review–revise: four rounds and 12 calls. Each round consists of three separate calls: a run call produces a complete candidate, a review call critiques it, and a revise call updates it using that review. The revised answer is carried into the next round. Four rounds therefore use 12 calls in total.

Iterative refinement: 13 sequential generations. The model produces an initial complete solution followed by 12 revisions. Each later call receives the article and current calculation and is asked to identify and correct errors or omissions. Each revision call both critiques and updates the answer. Both sequential protocols revisit the complete solution; the model chooses how to organize its derivation.

Measured token consumption. We pool the 93 positive and 93 negative instances in $\mathcal{I} = \{1, \ldots, 93\} \cup \{101, \ldots, 193\}$. For each model, B_i^s is strategy s's uncached token count on instance i, summed over all calls. We report its median $\widetilde{B}^{\,s}$ and the median instance-level ratio $\widetilde{r}^{\,s}$ to one-shot:

$$\widetilde{B}^{\,s} = \text{median}_{i\in\mathcal{I}}\, B_i^s, \qquad \widetilde{r}^{\,s} = \text{median}_{i\in\mathcal{I}} \left(\frac{B_i^s}{B_i^{\text{one}}} \right).$$

The one-shot median is 60,574 uncached tokens for DeepSeek v4 Flash and 47,030 for Qwen3.7 Plus. Table 2 shows that the expanded baselines use median ratios of $10.57\times$–$15.87\times$. Call counts are fixed; realized token use varies by strategy and instance. These budgets are calibrated to the pipeline's measured use across instances, rather than matched on each instance.

E.2 Evaluation and Aggregation

Completion on positive instances 1–93 uses R_{comp}, and responses to the 147 preset errors in negative instances 101–193 use R_{cov}. Both follow the assessment protocol in Appendix D.1. The sequential baselines are evaluated on their final revised outputs.

For parallel sampling, let $\mathcal{U}$ be the evaluation units and let $k_u \in \{0, \ldots, 13\}$ count successful outputs for unit u. On positive instances, $\mathcal{U}$ contains 93 targets and a success means that an output is judged complete. On negative instances, $\mathcal{U}$ contains 147 error annotations and a success is a `FOUND` judgment. We aggregate the 13 independent outputs at caps $h \in \{1, 3, 7\}$:

$$S^{(h)}_{\text{parallel}} = \frac{1}{|\mathcal{U}|} \sum_{u\in\mathcal{U}} \frac{\min(k_u, h)}{h}.$$

The cap $h = 1$ records whether any output succeeds. With $h = 3$, one, two, and at least three successes receive $1/3$, $2/3$, and full credit, respectively. The cap $h = 7$ similarly reaches full credit

at seven successes. All three scores lie in $[0, 1]$ and are reported as percentages. They use the same 13 outputs, with the evaluation unit fixed within each comparison.

On negative instances, $h = 1$ gives union coverage: different answers can contribute different annotated errors from the same article. Higher caps give more credit to errors detected repeatedly.

E.3 Enhanced Inference Baselines

The full API pipeline provides the reference for the three enhanced baselines defined in Appendix E.1.

Completion on positive instances. Table 9 reports the enhanced baselines' completion results on positive instances 1–93. Iterative refinement completes 6 of 93 targets for DeepSeek and 1 of 93 for Qwen; review–revise completes 9 and 0, respectively. The corresponding full API pipelines achieve completion rates of 92.5% and 65.6% (Table 1). At least one of the 13 parallel outputs completes the target on 8 of 93 instances for DeepSeek and 3 of 93 for Qwen. The $h = 3$ scores are 8.60% and 2.15%, respectively; the $h = 7$ scores are 7.83% and 1.08%.

Table 9: Enhanced-baseline completion on positive instances 1–93. Entries are percentages: $S^{(h)}_{\text{parallel}}$ for 13-output parallel sampling and R_{comp} for the sequential baselines' final outputs. The cap $h = 1$ measures at least one success; $h = 3$ and $h = 7$ award full credit at three and seven successes, respectively.

	R_{comp} or $S^{(h)}_{\text{parallel}}$ (%)	
Method	DeepSeek v4 Flash 0731	Qwen3.7 Plus
Parallel sampling ($h = 1$)	8.60	3.23
Parallel sampling ($h = 3$)	8.60	2.15
Parallel sampling ($h = 7$)	7.83	1.08
Iterative refinement (13 generations)	6.45	1.08
Review–revise (4 rounds)	9.68	0.00

Responses to injected defects. Table 10 reports annotation-level scores on negative instances 101–193. For each alternative, we report the gap to the full pipeline in percentage points. Parallel sampling uses $S^{(h)}_{\text{parallel}}$; the other methods use R_{cov}, as defined in Appendix E.2. Across the 13 outputs, DeepSeek flags 120 of the 147 preset errors at least once, and Qwen flags 121. These union coverages, 81.6% and 82.3%, are close to the full API pipeline's 83.0% and 88.4%, leaving gaps of 1.4 and 6.1 percentage points. The completion gains on positive instances are substantially larger.

At $h = 3$, the parallel-sampling scores are 71.4% for DeepSeek and 76.6% for Qwen; at $h = 7$, they are 65.9% and 69.6%. The decrease shows that some errors in the union are found in only a few outputs.

Among the sequential baselines evaluated with R_{cov}, iterative refinement has the highest coverage for both models. The full pipeline exceeds it by 19.1 percentage points for DeepSeek and 8.8 points for Qwen3.7 Plus. Removing the roadmap lowers Qwen's coverage from 88.4% to 83.0%, a difference of 5.4 points.

E.4 Pipeline Component Ablations

We evaluate four Qwen3.7 Plus API variants at xhigh effort: removing review–revision at multiple stages, constructing the dependency graph in one call, omitting the roadmap, and omitting proof extraction. Table 11 compares their bound completion with the full API pipeline. The full Codex configuration provides an execution-environment reference; both full configurations use the same pipeline and prompts (Appendix D.2). The single-call variant changes graph construction only. Each variant uses its own inference allocation.

Bound-completion ablations use positive instances and the protocol in Appendix D.1. Table 10 reports the roadmap's effect on negative-instance coverage. We also compare graph construction with graph review disabled, using workflow completion to measure whether a run reaches its final output.

Table 10: Annotation-level scores on negative instances 101–193. Δ is the full API pipeline's score minus the row's score, in percentage points. Parallel sampling uses $S^{(h)}_{\text{parallel}}$ over the same 13 outputs at caps $h \in \{1, 3, 7\}$; other rows use R_{cov}. A dash denotes an unreported result.

	DeepSeek v4 Flash		Qwen3.7 Plus	
Method	R_{cov} or $S^{(h)}_{\text{parallel}}$ (%)	Δ	R_{cov} or $S^{(h)}_{\text{parallel}}$ (%)	Δ
Full API pipeline	**83.0**	0.0	**88.4**	0.0
Parallel sampling ($h = 1$)	81.6	1.4	82.3	6.1
Parallel sampling ($h = 3$)	71.4	11.6	76.6	11.8
Parallel sampling ($h = 7$)	65.9	17.1	69.6	18.8
Iterative refinement (13 generations)	63.9	19.1	79.6	8.8
Review–revise (4 rounds)	61.2	21.8	76.2	12.2
Pipeline without roadmap	—	—	83.0	5.4

Table 11: Bound-completion ablations for Qwen3.7 Plus (xhigh). Rates are evaluated on positive instances using the completion protocol in Appendix D.1. Bold marks the highest reported rate.

Configuration	R_{comp} (%)
Full pipeline (Codex)	58.1
Full pipeline (API)	**65.6**
API without review–revision	60.2
API with single-call dependency graph	47.3
API without roadmap	57.0
API without proof extraction	54.8

Incremental dependency construction. The single-call graph variant has a bound-completion rate of 47.3%, 18.3 percentage points below the full API pipeline's 65.6%. Incremental construction repeatedly checks whether a node is atomic and traces its nearest proof dependencies before local quantitative resolution (Appendix B.3). This process exposes the quantitative premises that must be recovered and propagated to the target.

On positive instances 1–93, the workflow completion rate is $R_{\text{run}} = N_{\text{run}}/93$, where N_{run} counts runs that reach the final output. With graph review disabled in both configurations, incremental construction reaches a final output on 87 of 93 tasks (93.5%), compared with 63 of 93 (67.7%) for single-call generation, a difference of 25.8 percentage points. There are 29 tasks completed only by incremental construction and five only by single-call generation. Bound completion also requires a correct and complete proof, so $R_{\text{comp}} \leq R_{\text{run}}$ for each configuration, with both rates using all 93 tasks as the denominator.

Roadmap and target coverage. The roadmap establishes the target, its parameter regime, and the global proof route before dependency construction and local proof analysis. On instances 1–93, the no-roadmap run is judged complete for 53 of 93 tasks (57.0%) under the completion criterion above. A separate trace from negative instance 150 illustrates how the roadmap preserves numerical information.

Case study: retaining a numerical definition (Article 50). The sparse-graph query problem uses an exact-search subroutine with budget

$$A(N, r) = \left\lceil \frac{\pi}{8}\sqrt{\frac{N}{r}} \right\rceil + 4, \qquad N = \binom{n}{2}.$$

The full pipeline preserves this definition in the extracted proof and sums the subroutine costs. Its final $n^{3/2}$ bound has the explicit coefficient $\frac{\pi}{4}\sqrt{c/2} + 5c + 20\sqrt{2} + 60$, where $c \geq 1$ is the supplied sparsity certificate. The no-roadmap run leaves the sum $\sum_r A(N, r)$ unresolved and states that the

numerical definition is unavailable, although it is present in the same source document. The full pipeline carries this definition through its intermediate stages and uses it in the final calculation.

Review and revision. Removing review–revision at multiple stages lowers completion from 65.6% to 60.2%, a difference of 5.4 percentage points. The following trace shows a numerical error caught during review.

Case study: review repairs the numerical derivation (Article 25). For tropical circuits, the Qwen API draft incorrectly rewrites $3^{(n-2)/3}$ as $3^{n/3}/9$. Its review flags the equality and identifies the correct factor, $3^{-2/3}$. The final proof replaces that step with the explicit threshold inequality

$$n(8\ln 3 - 3\ln 2) \geq 16\ln 3,$$

which holds for every integer $n \geq 3$, and concludes $S_+(\Phi) \geq 2^{n/8}$, where $S_+(\Phi)$ counts the circuit's binary $+$ gates. The final coefficient was already conservative enough; review repaired the written justification supporting it.

Proof extraction. Without proof extraction, the API pipeline achieves a bound-completion rate of 54.8%, approximately 10.8 percentage points below the full pipeline's 65.6%. Extracting each local proof gives the quantitative-resolution stage a focused account of the argument and the premises needed to derive an explicit bound.

E.5 Completed One-shot Targets

Table 12 records the one-shot outputs that complete the requested target under the assessment protocol in Appendix D.1. Each configuration is evaluated on the 93 positive instances.

Table 12: One-shot completion on positive instances 1–93. Target IDs identify outputs judged complete under the common completion-assessment protocol.

Mode	Model	R_{comp} (%)	Target IDs
API	DeepSeek v4 Flash	5.38	26, 28, 52, 60, 89
API	Qwen3.7 Plus	1.08	74
Codex	DeepSeek v4 Flash	9.68	26, 28, 31, 52, 63, 74, 84, 87, 92
Codex	Qwen3.7 Plus	3.23	28, 74, 89

F Constant Recovery Across External Theorems

When processing papers that invoke external theorems, recovering hidden constants can require several levels of recursive expansion. With a small modification, ConstAble's DAG-based pipeline can handle these dependencies: for an external node with unresolved constants, the pipeline locates the required theorem or lemma in the cited paper, recursively invokes ConstAble, and substitutes the expanded bound back into the DAG with the appropriate parameter substitutions and assumptions. We conducted experiments on the papers of Lovitz & Lowe (2024) and Montanaro & Zhou (2024) to illustrate this mechanism (Figure 6).

Nearly tight bounds for testing tree tensor network states (Lovitz & Lowe, 2024). Theorem 2.4 uses the rank-testing bound from Lemma 6.2 of O'Donnell & Wright (2015). Applying ConstAble to that lemma gives the explicit sufficient copy count $N_{\mathrm{rank}} = \lceil 24r^2/\eta \rceil$ for a one-sided rank test with error at most $1/3$, where r is the rank bound and η is the trace-distance parameter. Substituting $\eta = \varepsilon^2/[2(n-1)]$ into the proof of Theorem 3.1 gives

$$N_{\mathrm{test}} \leq \left\lceil \frac{48(n-1)r^2}{\varepsilon^2} \right\rceil.$$

For $n \geq 2$, integer $r \geq 1$, and $0 < \varepsilon \leq 1$, this makes the upper-bound clause of Theorem 1.1 explicit.

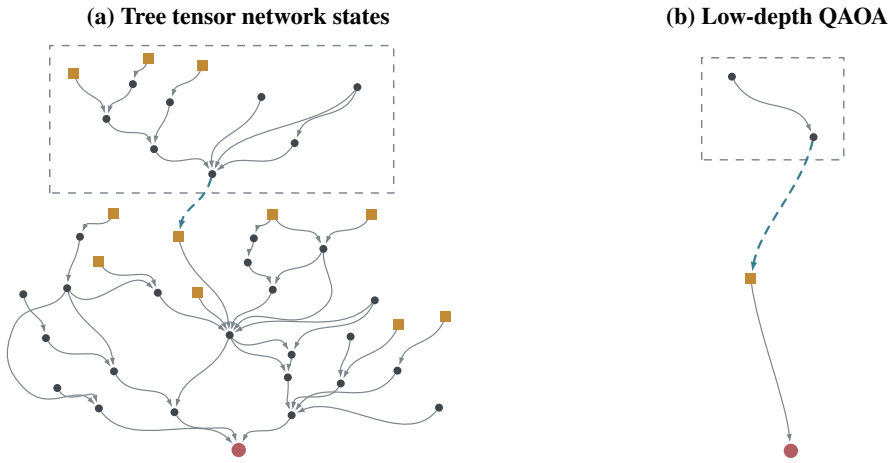


Figure 6: Cross-paper DAGs for (a) *Nearly tight bounds for testing tree tensor network states* (Lovitz & Lowe, 2024) and (b) *Quantum speedups in solving near-symmetric optimization problems by low-depth QAOA* (Montanaro & Zhou, 2024), generated by ConstAble with external-theorem handling. Dashed boxes enclose cited papers. Gray nodes denote proof units, yellow nodes external results, and red nodes the main targets. Dashed blue arrows show cross-paper substitutions; all arrows point from support to dependent.

Quantum speedups in solving near-symmetric optimization problems by low-depth QAOA (Montanaro & Zhou, 2024). The classical query upper bound in Claim 5.1 uses a quantitative group-testing result from Wang et al. (2016). Applying ConstAble to Theorem 3.1 and its supporting Lemma 4.1 gives $m_t = 3^t$ queries for $n_t = (t+3)3^{t-1}$ items. Padding to arbitrary $n \geq 2$ yields the explicit bound $m(n) \leq 12n/\log_3 n$. For the known cost function $C(x) = c(|x \oplus s|)$ with distinct values $c(0), \ldots, c(n)$, one initial query learns the hidden string's Hamming weight $|s|$, and each subsequent subset-count query requires one query to C. Substitution into Claim 5.1 therefore gives

$$Q_{\mathrm{cl}}(n) \leq 1 + \frac{12n}{\log_3 n} = 1 + 12\ln(3)\frac{n}{\ln n}, \qquad n \geq 2.$$

G Prompt Details

G.1 One-Shot Prompt

Table 13 reproduces the fixed one-shot instruction. Each instance supplies its own target and paper source.

Table 13: Complete fixed one-shot prompt.

Using only the target and paper source below, derive a self-contained, fully constant-expanded proof of the target bound. Strictly adhere to the paper's own proof strategy, logical structure, and sequence of derivations throughout the constant expansion. Do not access the internet.
Expand hidden constants, asymptotic notation, and qualitative size conditions into explicit inequalities, coefficients, thresholds, and parameter ranges, including relevant probability and error guarantees. Show the quantitative derivations and dependency substitutions, including how their assumptions are satisfied. Preserve explicit numerical information from the supplied material. Use exact numerical values when all inputs are fixed, and explicit expressions with admissible parameter dependence otherwise. Define any abbreviations; do not hide computable quantities behind unspecified constants or let a threshold depend on its own variable.
Do not introduce additional assumptions, external results, proof strategies, or arbitrary parameter restrictions. If the supplied material cannot determine a required quantity, state this rather than guessing. Clearly identify any unresolved external constants already present in the input and label conclusions depending on them as conditional rather than fully numerical.

G.2 Expanded Baseline Prompts

Parallel sampling uses Table 13 unchanged for each of 13 independent generations. Iterative refinement starts with the same instruction, then uses Table 14's follow-up for 12 revisions. Review–revise applies the three instructions in Table 15 for four rounds. Both sequential protocols retain Table 13's requirements.

Table 14: Iterative refinement follow-up prompt.

Revise the previous draft under the requirements above. Treat the draft as an unverified candidate, not as evidence. Correct any unsupported claims or errors you identify, and return a complete revised answer.

Table 15: Review–revise prompts, used in each of four rounds.

Run Produce a complete working draft satisfying the requirements above. If a previous-round revised answer is supplied, use it only as an unverified starting point and reconsider its claims against the target and paper. Return the complete draft.
Review Review the candidate proof below against the target, paper source, and requirements above. Do not assume the candidate is correct. Identify unsupported claims, incorrect derivations, missing dependencies, unexpanded constants, circular thresholds, and unsatisfied assumptions or probability/error conditions. Distinguish source-supported corrections from quantities the supplied material cannot determine. Give concrete corrections justified by the supplied source. Do not access the internet or introduce external results. Return the review rather than a rewritten proof.
Revise Revise the candidate proof using the review below, while checking every proposed correction against the target and supplied paper source. Treat both the candidate and the review as unverified material. Preserve the paper's own proof strategy and obey all requirements above. Do not introduce unsupported assumptions, external results, or guessed constants. Return a complete revised proof, not a change list or a response to the reviewer.

G.3 Pipeline Prompt Protocol

Table 16 gives the central instructions of the four pipeline prompt families. Their complete text for both Codex and API runs is included in the supplementary materials.

Table 16: Core requirements of the four pipeline prompt families.

Stage	Core prompt requirement
Roadmap	Locate the selected bound in the paper, preserve its formal scope, and identify the minimal setting and proof route needed for that bound.
Dependency graph	Use the paper TeX to trace direct, load-bearing proof dependencies; keep definitions and assumptions in a separate registry rather than treating them as proof nodes.
Proof extraction	Extract the paper-local proof of each target node, use direct dependencies as established results, and report when the paper provides no local proof.
Hidden-constant resolution	Expand constants, thresholds, parameter conditions, and probability guarantees through the local proof and its dependencies. Preserve source-explicit numbers and identify unresolved external constants.