AI DISCOVERY IN COMBINATORICS: FROM PROBLEMS TO SOLUTIONS AND BACK TO BIGGER PROBLEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce qtBench, a benchmark of 29 open problems in algebraic combinatorics. These problems ask for a simple combinatorial algorithm that explains given polynomial coefficients or identities. To help AI agents search for such algorithms, we provide a scaffold for playing with combinatorial objects, countering cheating attempts, and testing candidate solutions. To explore how far frontier AI systems can go on these problems, we collaborated with OpenAI, whose researchers obtained proposed solutions to several of them. This highlights bigger meta-problems: is AI discovery at scale ethical? How should these results be released? How should proof digestion be organized? We discuss these questions and give our views.

1 INTRODUCTION

A recurring problem in science and mathematics is to induct a simple deterministic rule that fully explains observable data. The motion of celestial bodies is a famous example: patterns in planets' trajectories led Newton to a compact law of gravitation, capable of explaining a much richer collection of observations. Mathematics has its own instances of this class of problems. Consider the problem of counting labeled trees on n vertices. Direct enumeration for $n = 2, 3, 4, 5, 6, \ldots$ produces the seemingly unstructured sequence $1,\ 3,\ 16,\ 125,\ 1296, \ldots$ Yet Cayley's formula reveals that these numbers are exactly n^{n-2}. A collection of observations is thus explained perfectly by a simple rule. The focus of this paper is on mathematical problems of this kind, where approximating or fitting the observed data is simply wrong: a solution must discover a *simple* deterministic algorithm that explains the observations *exactly*.

As large language models (LLMs) and LLM-based agents become increasingly capable at mathematical and scientific reasoning, it is important to understand to which degree they can perform this kind of discovery. Frontier systems are now able to make meaningful progress on open research questions: recent successes include improving state-of-the-art bounds (Georgiev et al., 2025), extending or completing arguments whose main structure is already present in the literature (Diez et al., 2025), and discovering counterexamples to long-standing conjectures (Alon et al., 2026). In several such settings, progress can be incremental: a better bound is already a contribution even if it is not optimal, and improvement to one step of an existing argument can be mathematically valuable on its own. The problems we consider here are different. The AI system is given data generated by an unknown deterministic rule and must induct the rule itself. There is no incremental contribution: a rule either reproduces the data exactly or it does not. Success may require forming hypotheses, testing them computationally, recognizing when an apparently promising direction contains no path to an exact solution, abandoning it, and stepping back to tackle the problem from a different perspective. We seek to understand how current models behave in this regime, which simultaneously demands *long-horizon reasoning* and *creative exploration*, while being able to *verify* proposed solutions efficiently.

Problems from algebraic combinatorics, and in particular from q, t-combinatorics, provide an unusually clean setting in which to study these capabilities. Many such problems arise from a recurring phenomenon in algebra and representation theory: one encounters an infinite family of polynomials indexed by natural numbers, say $n \in \mathbb{N}$, defined in an algebraic or representation-theoretic way.

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.

When these polynomials are computed explicitly for small values of n (e.g. $n \leq 7$), a clear pattern emerges: their coefficients are non-negative integers, for reasons that are not immediately apparent from the original definition. Then two natural questions arise: are the coefficients natural for all $n \in \mathbb{N}$, and, if so, what combinatorial objects do these coefficients count? A combinatorial interpretation answers both questions at once: it explains why the coefficients are non-negative integers and gives them a concrete meaning. Discovering such interpretations, and finding combinatorial explanations for identities among the resulting polynomials, is the focus of the problems considered in this paper.

With these classes of problems, we present qtBench, a benchmark of 29 open problems and 1 solved calibration problem in q, t-combinatorics. Our framework is defined by four key contributions:

- We present an open-source benchmark of 30 problems across q-statistic discovery, t-statistic discovery, and bijection discovery. Because the solutions are unknown (except for the single calibration problem)[1], they do not exist in any training corpus, and any correct solution produced by a model would therefore signal genuine reasoning ability and autonomous discovery.
- We provide an agent scaffold for attempting these problems: for each one, models receive a code implementation of the combinatorial objects together with the target polynomials for a range of parameters. We also develop an automatic evaluation pipeline that checks whether a proposed statistic or bijection satisfies the required combinatorial identities and detects several forms of cheating.
- We present a baseline evaluation of GPT-6 Astra (High), with two hours per problem using our inference scaffold, together with results from a collaboration with OpenAI on evaluating their internal models. These results include promising candidate solutions that, if verified, would constitute valuable contributions to algebraic combinatorics.
- Motivated by these results, we discuss the ethical questions raised by AI-driven math discovery at scale and present our perspective on how such research should be conducted.

2 Related Work

2.1 Benchmarks of Open Research Problems in Mathematics

As AI's mathematical capabilities grow, benchmarks are needed to measure what these systems can do and where they still fall short. Recent benchmarks have begun to test them on open research problems. HorizonMath (Wang et al., 2026) contains mostly unsolved problems from eight areas of mathematics, with code to check proposed closed-form expressions, improvements to known bounds, and new constructions. FrontierMath: Open Problems (Epoch AI, 2026) also focuses on open problems whose proposed solutions can be checked computationally, using a program written for each problem. Formal Conjectures (Firsching et al., 2026) instead provides statements formalized in Lean 4, including both open conjectures and solved problems, so that proposed proofs can be checked by the proof assistant. Recent AI-assisted work has also drawn attention to Thomas Bloom's Erdős Problems database (Bloom, 2026), which collects problems posed by Erdős and records progress towards their solution. Our benchmark contributes to this line of work by focusing on open research problems, specifically in the field of algebraic combinatorics.

2.2 AI for Discovering Combinatorial Statistics and Bijections

Recently, researchers have begun using AI to search for combinatorial statistics and bijections. Huang et al. (2025) trained a Transformer on pairs of Dyck paths and used its attention patterns to find a new algorithm for the zeta map (Haglund, 2008). Brown et al. (2025) used OpenEvolve, an open-source system that combines LLM-generated programs with evolutionary search, to search for bijections in three problems involving Dyck paths. They recovered one known bijection but did not solve the open area-bounce problem they studied. Cainelli et al. (2026) used machine learning to discover statistics for q, t-Narayana polynomials, leading to an explicit bijection proving the q, t-symmetry of $N_{n,3}(q, t)$. The studies above use either ad-hoc models or LLMs embedded in an

[1] While preparing this paper, problems 7 and 17 were solved and their solutions published on arXiv.

evolutionary search procedure. In the present work, we investigate whether general-purpose LLM agents can make progress when given tools for exploring combinatorial objects and testing candidate solutions, while leaving them free to choose their own search strategy.

3 q,t-COMBINATORICS SETUP

We study problems that ask for combinatorial interpretations of polynomial coefficients and identities. We explain the three resulting types of discovery problems through a single running example. A *Dyck path* of semilength n is a lattice path from $(0,0)$ to (n,n), using north and east unit steps, that stays weakly above the main diagonal. We write Dyck_n for the set of such paths. These objects are counted by the Catalan number $\mathrm{Cat}_n = \frac{1}{n+1}\binom{2n}{n}$.



area word 01212011 (a) $\mathrm{area}(D) = 8$

bounce word 00011223 (b) $\mathrm{bounce}(D) = 9$

Figure 1: An 8×8 Dyck path. In (a), each label on the left counts the shaded cells between the path and the main diagonal in that row. Read from bottom to top, these counts form the area word 01212011, whose entries sum to $\mathrm{area}(D) = 8$. In (b), the dashed line is the bounce path, and the labels on the left give its vertical-step labels, again read from bottom to top. They form the bounce word 00011223, whose entries sum to $\mathrm{bounce}(D) = 9$.

3.1 q-STATISTIC DISCOVERY

A q-statistic discovery problem starts with an infinite family of polynomials $F_n(q)$. As our running example, take $F_n(q)$ to be the q-Catalan polynomials (Carlitz & Riordan, 1964), which arise naturally in algebraic combinatorics. Setting $q = 1$, we obtain $F_n(1) = \mathrm{Cat}_n$, the ordinary Catalan number. Since Catalan numbers count Dyck paths, this suggests that the full polynomial might refine the ordinary count.

The task is to find a natural function (called statistic)

$$\mathrm{stat} : \mathrm{Dyck}_n \longrightarrow \mathbb{N} \qquad \text{such that} \qquad F_n(q) = \sum_{D \in \mathrm{Dyck}_n} q^{\mathrm{stat}(D)}$$

for every n. If this equality holds, then the coefficient of q^k has a concrete meaning: it is exactly the number of paths D for which $\mathrm{stat}(D) = k$. Finding such a statistic therefore explains why every coefficient of $F_n(q)$ is a non-negative integer.

For the q-Catalan polynomials, one solution is the *area* statistic. It counts the whole lattice cells between the path and the main diagonal, as in Figure 1, and satisfies

$$F_n(q) = \sum_{D \in \mathrm{Dyck}_n} q^{\mathrm{area}(D)}.$$

For example, $F_3(q) = 1 + 2q + q^2 + q^3$: among the five Dyck paths of semilength 3, one has area 0, two have area 1, one has area 2, and one has area 3.

The word *natural* is important. If the polynomial $F_n(q)$ has coefficients $a_0, a_1, \ldots$, one could arbitrarily partition Dyck paths into groups of sizes $a_0, a_1, \ldots$ and assign value i to every path in the

group of size a_i. The resulting statistic would reproduce the polynomial by construction, without explaining why the family of polynomials arises from these objects. The goal is a simple rule or algorithm that depends on the structure of each object and works uniformly for the whole infinite family. There may be several such rules and the task is to discover any of them.

3.2 t-STATISTIC DISCOVERY

A t-statistic discovery problem is similar, but starts from an infinite family of polynomials in two variables, $F_n(q,t)$. Part of the combinatorial interpretation is already known: we are given a family of objects and a statistic qstat satisfying

$$F_n(q,1) = \sum_{D \in \mathrm{Dyck}_n} q^{\mathrm{qstat}(D)}.$$

What is missing is a natural partner tstat such that

$$F_n(q,t) = \sum_{D \in \mathrm{Dyck}_n} q^{\mathrm{qstat}(D)} t^{\mathrm{tstat}(D)}.$$

The coefficient of $q^i t^j$ counts how many paths have $(\mathrm{qstat}, \mathrm{tstat}) = (i,j)$. Setting $q = 1$ forgets the known statistic and tells us only how many paths must receive each value of the missing statistic. To match the full polynomial $F_n(q,t)$, these counts must also be correct within each group of paths having the same value of qstat.

The q-Catalan polynomials have a two-variable refinement, the q,t-Catalan polynomials $C_n(q,t)$. This more general family recovers the one-variable polynomials when either variable is set to 1:

$$C_n(q,1) = F_n(q) \qquad \text{and} \qquad C_n(1,t) = F_n(t).$$

Therefore, we can take area as the known statistic and $C_n(q,t)$ as the target polynomial. The problem becomes to find a second statistic tstat such that

$$C_n(q,t) = \sum_{D \in \mathrm{Dyck}_n} q^{\mathrm{area}(D)} t^{\mathrm{tstat}(D)}.$$

One natural solution is Haglund's *bounce* statistic Haglund (2003). Its discovery took eight years of sustained community effort and was a major breakthrough. To define bounce, associate with a Dyck path D a new lattice path, called the *bounce path*. Beginning at $(0,0)$, it moves north until it reaches the initial point of an east step of D, and then it moves east until it returns to the main diagonal. This procedure is repeated until the path reaches (n,n). Its vertical steps are labelled starting from 0, with the label increasing by 1 after each return to the diagonal. The bounce statistic is the sum of these labels. An example is given in Figure 1. A different statistic, *dinv*, is defined using certain pairs of steps, and it gives another solution to the same problem. Thus bounce and dinv are natural statistics defined in different ways, but both, together with area, explain the same q,t-Catalan polynomials.

3.3 BIJECTION DISCOVERY

A bijection discovery problem starts from a situation in which the pair of relevant statistics are already known, but we do not understand why their joint distribution has a specific property. For instance, the q,t-Catalan polynomial is symmetric:

$$C_n(q,t) = C_n(t,q).$$

Since $C_n(q,t) = \sum_D q^{\mathrm{area}(D)} t^{\mathrm{bounce}(D)}$, this symmetry implies that, for every pair of nonnegative integers $i,j \in \mathbb{N}$, there are as many Dyck paths with statistics $\mathrm{area}(D) = i$ and $\mathrm{bounce}(D) = j$ as there are paths with statistics $\mathrm{area}(D) = j$ and $\mathrm{bounce}(D) = i$. The natural question is why.

To explain this combinatorially, we would like to find a bijection $\phi : \mathrm{Dyck}_n \to \mathrm{Dyck}_n$ satisfying, for every path D,

$$\mathrm{area}(\phi(D)) = \mathrm{bounce}(D), \qquad \mathrm{bounce}(\phi(D)) = \mathrm{area}(D). \tag{1}$$

Such a bijection gives a direct combinatorial proof of the q, t-symmetry: every object contributing to the coefficient of $q^i t^j$ is sent to an object contributing to the coefficient of $q^j t^i$. Although the symmetry of q, t-Catalan polynomials is known by algebraic methods, no explicit bijection exchanging area and bounce is known; finding one is an important, decades-old open problem with possible implications beyond combinatorics, and one of the problems in qtBench.

Again, an arbitrary bijection satisfying (1) is not a solution. Once symmetry is known, one can order the paths with statistics (i, j) and match them to paths with statistics (j, i). Such a construction assumes the symmetry it is supposed to explain and gives no combinatorial insight. The goal instead is an explicit algorithm that explains the symmetry.

4 Benchmark Design

The goal of qtBench is to discover conjectural statistics and bijections, without requiring proofs of correctness. This reflects a common practice in algebraic combinatorics: a natural algorithm supported by extensive numerical evidence is taken seriously as a conjecture and provides solid grounds for expecting the required identity to hold in general for the whole infinite family of polynomials. Finding such a conjectural function is a substantial mathematical contribution in its own right. Proving it correct is a separate problem that may require significant further effort.

4.1 Design Principles

qtBench follows four design principles. First, every problem must belong to one of the three discovery categories introduced in Section 3: q-statistic discovery, t-statistic discovery, or bijection discovery. These three categories are broad enough to cover many important problems in algebraic combinatorics. Second, each problem must be mathematically meaningful. It is either stated as an open problem in the literature or arises naturally from established combinatorial objects, polynomials, or identities, and may already have been posed informally within the community. Third, except for the solved calibration problem, every problem must be open. Since no accepted solution is available, a successful result provides evidence of an AI agent's capability for this kind of discovery and could itself contribute to algebraic combinatorics. Fourth, we must be able to compute the target polynomials for a sufficiently wide range of parameter values to support the search for a solution. The first few cases may reveal too little structure for an agent to form and test meaningful hypotheses. Generating these data should take at most tens of minutes of CPU time on a machine with 64 GiB of RAM, while checking a proposed solution should take less than ten minutes.

4.2 Curation Procedure

We identify candidate problems through a manual review of the literature in q, t-combinatorics and through discussions with experts in the field. For each candidate, we identify the main references defining the relevant objects, polynomials, or identities. We then verify, to the best of our knowledge, that no solution of the problem is known. We describe the relevant literature precisely. Finally, we implement the relevant combinatorial objects and known statistics and compute the given polynomials for a finite range.

qtBench contains 30 problems: 29 open problems and 1 solved calibration problem. They are organized by discovery category, as summarized in Table 1.

Table 1: The 30 problems of qtBench by discovery category.

Category	**Problem description**	**Count**
q-statistic discovery	Given a combinatorial object and a one-variable polynomial family, find a statistic that generates the polynomial.	8
t-statistic discovery	Given a combinatorial object, one (sometimes none) known statistic, and a multivariate polynomial, find the missing statistic(s) that generates the polynomial together with the known statistic (if available).	13
bijection discovery	Given a combinatorial object and two statistics, find a bijection on the set of combinatorial objects satisfying the required identity.	9

4.3 Problem Package

Each problem comes with a self-contained statement describing the combinatorial objects, how they are represented in Python, the known statistics, and the target polynomials or identity to be explained. We provide references to the relevant literature and examples of the combinatorial objects, together with their known statistic values. The agents can use these examples and the code for the known statistics to explore the problem and test new ideas. Each statement specifies which Python functions to submit and provides a command for evaluating candidate solutions. Most statistic problems ask for `statistic(object)`, while bijection problems ask for `forward(object)` and `inverse(object)`. We describe the scaffold for the evaluation in the next section.

5 Automated Evaluation

5.1 Numerical Verification

A proposed statistic can be tested by evaluating it on the combinatorial objects and checking whether it reproduces the target polynomials over the finite range for which they have been computed. For example, for a t-statistic discovery problem with known statistic qstat and submitted statistic tstat, the checker computes

$$\widehat{F}_n(q,t) = \sum_{D \in \mathrm{Dyck}_n} q^{\mathrm{qstat}(D)} t^{\mathrm{tstat}(D)} \tag{2}$$

over the same range of n as the given ground-truth polynomials $F_n(q,t)$, then checks that $\widehat{F}_n(q,t) = F_n(q,t)$ for every n in this range.

For bijection discovery, we check that `forward` and `inverse` return valid objects, are mutually inverse, and satisfy the identities required by the problem on the objects of the scored range.

5.2 Cheating Attempts

Even if a submission passes numerical verification over the entire given range, this alone does not tell us whether it has overfit the computed examples or provides a meaningful solution to the problem. As discussed in Section 3, arbitrary assignments generate the target polynomials or identities while assuming the very property a combinatorial interpretation should prove. We call such functions *cheating attempts*.

In discussions with experts in the field, we found no formal criterion for what makes a statistic or bijection a meaningful combinatorial construction, or what should be instead classified as a cheating attempt. A proof formalized in Lean or a similar proof assistant could certify the required identities for all $n \in \mathbb{N}$, but would not by itself establish that the combinatorial problem is solved. Determining whether the submission solves the problem requires mathematical judgment. In practice, a mathematician can distinguish a natural statistic from an arbitrary assignment by inspecting its definition, even without a general formal criterion. In principle, stricter mechanical checks could reject all cheating attempts, but they could also exclude meaningful solutions. We therefore choose to build a checker whose acceptance is a necessary but not sufficient condition for a solution, and then use an LLM judge to assess whether the construction is natural or a cheating attempt. Our tests suggest that frontier models can make this distinction quickly and reliably (as described in the Appendix A), motivating this design choice. Even so, a submission that passes both the checker and the LLM judge is only a *candidate* solution, since expert mathematical review is always required.

5.3 Checker Design

We ask for short, self-contained Python submissions and evaluate them using the same five checker stages for every problem. The stages run in order, and evaluation stops at the first failure. Submissions that pass all five stages proceed to the LLM judge, which assesses whether they provide a meaningful combinatorial construction. Those approved by the judge should still be reviewed by an expert mathematician (Figure 2).

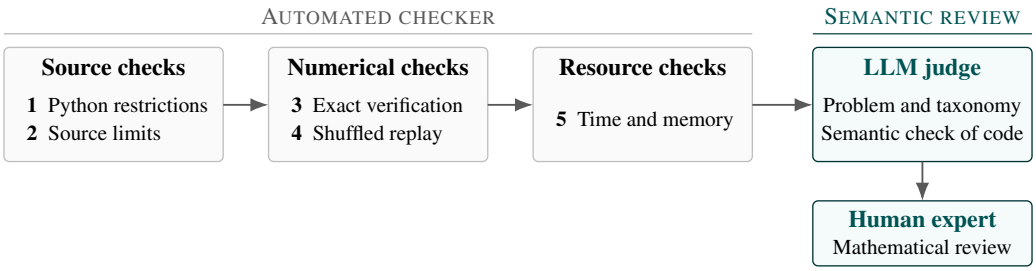

Figure 2: Evaluation pipeline for benchmark problems.

1 and 2. Source checks. We first inspect the submitted code without running it. For both statistic and bijection discovery, we require a single, self-contained Python file. Submissions may define helper functions and use loops, recursion, and built-in data structures. They may not import modules or define classes, and they may use only a small set of Python's built-in functions. The first check, a *capability screen*, enforces these rules and also rejects code that accesses files, uses reflection, or reaches into Python's runtime internals.

The second check limits the size of the source code. Since we expect statistics and bijections to admit short implementations, we apply the same limits to both task types: at most 8,192 bytes, 160 lines, 1,500 tokens (excluding comments and whitespace), 2,000 nodes in the abstract syntax tree, and 256 bytes of literal constants in total. These limits are intended to exclude submissions that store precomputed answers instead of implementing a combinatorial rule, and to keep submissions easy to review.

3 and 4. Numerical checks. We then run the submission on the finite collection of combinatorial objects selected for the problem. For statistics, we check that the resulting polynomials equal the supplied target polynomials. For bijections, we check that the outputs are valid objects, that the composition of the forward and inverse maps is equal to the identity map, and that the required identities hold.

Matching the target polynomials does not ensure that the submitted function assigns a deterministic value to each object: a program could distribute values according to the order in which objects are received. We therefore reload passing submissions and evaluate the same objects in a randomly shuffled order, with repeated calls. Each object must produce the same output as before. This detects dependence on call order or previous evaluations, but does not exclude arbitrary assignments that consistently associate the same output with each object.

5. Resource checks. We use time and memory limits to reject submissions that are too computationally expensive to evaluate. The numerical checks must finish within 60 seconds for statistics and 480 seconds for bijections.

5.4 Semantic Review and Acceptance

We maintain a taxonomy of cheating attempts that describes how they work, which checks detect them, and which attempts can still pass the checker. To expand this taxonomy, we conducted adversarial experiments in which LLMs were asked to produce submissions that pass the checker without solving the underlying combinatorial problem. These experiments helped us identify more cheating strategies and document the limitations of the checker.

The LLM judge, running in an isolated sandbox, receives the taxonomy together with the problem statement and the submitted code. It uses this context to assess whether the submission provides a meaningful combinatorial construction or constitutes a cheating attempt. The taxonomy informs this assessment but does not exhaust the possible ways to cheat. In our tests, the LLM judge distinguishes meaningful proposals from cheating attempts with high accuracy; we present the evaluation and its results in the Appendix A.

A submission is accepted as a candidate only if it passes all checker stages and receives the LLM judge's approval.

Getting agents to find genuine solutions is not only a mathematical problem, but also an alignment problem. When the models cheat, it is neither for lack of context, since the statements and our instructions spell out what counts as a solution and what counts as cheating, nor for lack of understanding, since frontier models reliably make this distinction when we use them as judges. They cheat because they are not well aligned. This applies to both GPT-6 Astra (High) and OpenAI's internal models: cheating was common in our baseline runs, and at least 4 of the 20 solutions proposed by OpenAI's models rely on some form of cheating. In our baseline evaluation, the reasoning traces exhibit a clear shift: it happens frequently that once a problem is deemed too hard, the model stops working on it and tries to game the verifier.

6 AI Discovery at Scale, Responsible Release and Community Effort

We created this benchmark because we thought that publicly available models would be unable to solve any of these open problems, with their tendency towards "low-entropy" thinking being the main capability bottleneck. In fact, in our experiments, which consisted in two hours of inference on each problem using GPT-6 Astra (High), no proposed solution has passed both the checker and the LLM judge.

We also began a collaboration with OpenAI to test their internal models and inference scaffold on these problems. We did not expect notable results, but OpenAI returned proposed solutions to 20 problems in the benchmark, 10 of which we consider genuine solutions to the given problems (see Appendix B). These include a bijective proof of the m, n-symmetry of the q, t-Narayana polynomials (Aval et al., 2014) and counterexamples to 3 positivity conjectures. We also received a proposed bijection for the symmetry of the q, t-Catalan polynomials, guided by a known algebraic proof (Carlsson & Mellit, 2018), which we still need to assess.

The LLM-generated material is poorly written and requires substantial work to understand and rewrite the solutions. This work is akin to what Tao (2026) calls *proof digestion*. Given the importance of these problems to algebraic combinatorics, and the increasing capabilities of LLMs in mathematics, we would like to open a discussion around the following questions, and give our own views in the rest of this section:

1. Is it ethical to use AI to solve a substantial proportion of a field's open problems?
2. How should the results be released?
3. How should the work of proof digestion be organized?
4. How should credit be allocated?

6.1 Ethics of AI-Assisted Discovery

What is mathematics, and what is its goal? Neither question has an easy answer. Yet, in these turbulent times, much of the mathematical community seems to be returning to the view expressed by William Thurston in *On proof and progress in mathematics* (Thurston, 1994): mathematical progress is about advancing human understanding, not just producing proofs. We take clarity and understanding to be a primary goal of mathematics throughout this section. With this goal in mind, is it good to use AI to prove results in a field at scale?

We believe that the new information supplied by AI in the form of proofs and counterexamples is neither inherently good nor inherently bad. Its value depends on how the community works with it and on the incentives surrounding that work. Faced with AI discovery at scale, we believe there is a set of trajectories of choices the community can follow that would have a net positive impact on the clarity and understanding of a field. Does leaving its traditions and incentive system unchanged belong to any of these trajectories? We do not think so. The community needs to change radically to respond to the increasing capabilities of AI. For the changes we support, we refer to Daniel Litt's essay *A beginning for mathematics* (Litt, 2026) and to Timothy Gowers's recent blog post (Gowers, 2026), which largely reflect our views on policy changes for the mathematical community.

The community cannot avoid these questions simply by calling for mathematicians to stop using AI. We expect tools with capabilities comparable to those now available internally at OpenAI to become widely accessible in the near future, as much as computers are available today. Once these tools are widely available, it would be unrealistic to enforce a blanket ban on their use, even within academia. Hence, the community needs to decide how to engage with AI-generated discoveries and turn them into shared mathematical understanding.

OpenAI themselves started an independent advisory group (OpenAI, 2026) to discuss these important questions; we look forward to seeing the outcome.

6.2 Release of Results

We believe it is fair to give the community access to AI-generated proposed solutions, since we are not able to digest all of them. At the same time, we do not want to flood arXiv, or the wider mathematical discussion, with poorly written AI-generated papers. We therefore propose to share them with researchers in the field that are interested in the digestion work, in writing a paper on the material they are able to digest, and in disseminating their understanding through talks at conferences.

We cannot control how these documents are used after their release. We therefore advocate for the community to digest the solutions before building on the results. Skipping this step risks piling poorly understood papers on top of one another, taking us further from the clarity and understanding we seek to achieve.

6.3 Proof Digestion

Part of the backlash against AI-generated solutions to important mathematical problems concerns the failure to properly acknowledge earlier work (see, e.g., Binder, 2026). When insights developed by mathematicians are presented without due credit, we believe the criticism is justified. An important part of proof digestion is therefore to establish which ideas are new and which are already present in the literature, even when the AI-generated text does not acknowledge them.

Once the LLM-generated documents are released, we suggest that groups interested in digesting a particular result discuss their plans openly, much as the Mathlib community coordinates work on Zulip. When a group has completed this work, its members can publish a paper presenting the digested result, stating in the acknowledgments or AI disclosure that their work builds on an AI-generated discovery. New results can then build on the digested proofs, with and without AI.

Should AI-generated results also be presented at conferences and workshops, as human-generated results are? We believe they should. Sharing new mathematics is essential to the progress of the field, regardless of whether AI contributed to its discovery. Worthy results, ideas, and techniques need to be understood by the community and become part of its mathematical toolkit. We believe that the people who digested the solutions and wrote the resulting papers are best placed to present this work and have earned that role through their efforts.

6.4 Credit and Attribution

We cannot dictate how credit should be allocated, but we highlight three main types of contribution applicable to the qtBench story: **problem selection, scaffold building, and inference**, the contribution described in this paper; **AI training**, the work of the company that trained the models; and **digestion and dissemination**, the work of those who digest the solutions, write the resulting papers, and explain the results to the community.

We believe the mathematical community should place increasing importance on the first and third contributions: choosing questions worth pursuing and turning their answers into shared understanding. Problem solving might never be fully automated, but advances in AI may soon decrease its relative importance in mathematical research. However, even AI that can discover, prove, and explain new mathematics does not remove the need for people to understand it. The future of mathematics depends not only on what machines can discover, but on what we humans come to understand.

AI USE STATEMENT

We used AI tools to help implement the benchmark and evaluation code, generate and assess candidate solutions, create the synthetic submissions used in Appendix A, and edit the paper. We manually reviewed the synthetic submissions and their labels. We take full responsibility for the paper's content.

ETHICS STATEMENT

Section 6 discusses the ethics of our work.

REPRODUCIBILITY STATEMENT

The supplementary material contains the benchmark code and data; installation and evaluation instructions are provided in `REVIEWER.md`.

For Problem 3, the code behind the target polynomials with $n \geq 5$ is not included, and the targets for $n = 7, 8, 9$ are conditional on some conjectures on the type B q, t-Catalan polynomials, not computed directly. Our baseline evaluation uses a publicly available model, GPT-6 Astra (High), although reruns may give different outputs; OpenAI's internal runs are not reproducible.

REFERENCES

Noga Alon, Thomas F. Bloom, W. T. Gowers, Daniel Litt, Will Sawin, Arul Shankar, Jacob Tsimerman, Victor Wang, and Melanie Matchett Wood. Remarks on the disproof of the unit distance conjecture. *arXiv preprint arXiv:2605.20695*, 2026. doi: 10.48550/arXiv.2605.20695. URL https://arxiv.org/abs/2605.20695.

Jean-Christophe Aval, Michele D'Adderio, Mark Dukes, Angela Hicks, and Yvan Le Borgne. Statistics on parallelogram polyominoes and a q, t-analogue of the Narayana numbers. *Journal of Combinatorial Theory, Series A*, 123(1):271–286, 2014. doi: 10.1016/j.jcta.2013.09.001. URL https://arxiv.org/abs/1301.4803.

Ilia Binder. After Leiden. *CMS Notes*, 58(4), 2026. URL https://notes.math.ca/en/article/after-leiden/.

Thomas Bloom. Erdős Problems, 2026. URL https://www.erdosproblems.com/. Accessed September 18, 2026.

Davis Brown, Jesse He, Helen Jenne, Henry Kvinge, and Max Vargas. Even with AI, bijection discovery is still hard: The opportunities and challenges of OpenEvolve for novel bijection construction. *arXiv preprint arXiv:2511.20987*, 2025. doi: 10.48550/arXiv.2511.20987. URL https://arxiv.org/abs/2511.20987.

Eugenio Cainelli, Lorenzo Luccioli, Alessandro Iraci, Michele D'Adderio, and Giovanni Paolini. Mapping uncharted symmetries: Machine discovery in combinatorics. *arXiv preprint arXiv:2605.19063*, 2026. doi: 10.48550/arXiv.2605.19063. URL https://arxiv.org/abs/2605.19063.

Leonard Carlitz and John Riordan. Two element lattice permutation numbers and their q-generalization. *Duke Mathematical Journal*, 31(3):371–388, 1964. doi: 10.1215/S0012-7094-64-03136-9. URL https://doi.org/10.1215/S0012-7094-64-03136-9.

Erik Carlsson and Anton Mellit. A proof of the shuffle conjecture. *Journal of the American Mathematical Society*, 31(3):661–697, 2018. doi: 10.1090/jams/893. URL https://arxiv.org/abs/1508.06239.

Charles-Philippe Diez, Luis da Maia, and Ivan Nourdin. Mathematical research with GPT-5: a Malliavin–Stein experiment. *arXiv preprint arXiv:2509.03065*, 2025. doi: 10.48550/arXiv.2509.03065. URL https://arxiv.org/abs/2509.03065.

Epoch AI. FrontierMath: Open problems, 2026. URL https://epoch.ai/frontiermath/open-problems. Accessed September 18, 2026.

Moritz Firsching, Paul Lezeau, Salvatore Mercuri, Miklós Z. Horváth, Yaël Dillies, Calle Sönne, Eric Wieser, Fred Zhang, Thomas Hubert, Blaise Agüera y Arcas, and Pushmeet Kohli. Formal conjectures: An open and evolving benchmark for verified discovery in mathematics. *arXiv preprint arXiv:2605.13171*, 2026. doi: 10.48550/arXiv.2605.13171. URL https://arxiv.org/abs/2605.13171.

Bogdan Georgiev, Javier Gómez-Serrano, Terence Tao, and Adam Zsolt Wagner. Mathematical exploration and discovery at scale. *arXiv preprint arXiv:2511.02864*, 2025. doi: 10.48550/arXiv.2511.02864. URL https://arxiv.org/abs/2511.02864.

Timothy Gowers. Why I didn't sign the Fields medallists' letter, 2026. URL https://gowers.wordpress.com/2026/09/17/why-i-didnt-sign-the-fields-medallists-letter/.

James Haglund. Conjectured statistics for the q, t-Catalan numbers. *Advances in Mathematics*, 175 (2):319–334, 2003. doi: 10.1016/S0001-8708(02)00061-0. URL https://doi.org/10.1016/S0001-8708(02)00061-0.

James Haglund. *The q, t-Catalan Numbers and the Space of Diagonal Harmonics*, volume 41 of *University Lecture Series*. American Mathematical Society, Providence, RI, 2008. doi: 10.1090/ulect/041. URL https://doi.org/10.1090/ulect/041.

Xiaoyu Huang, Blake Jackson, and Kyu-Hwan Lee. From black box to bijection: Interpreting machine learning to build a zeta map algorithm. *arXiv preprint arXiv:2511.12421*, 2025. doi: 10.48550/arXiv.2511.12421. URL https://arxiv.org/abs/2511.12421.

Daniel Litt. A beginning for mathematics, 2026. URL https://www.daniellitt.com/blog/2026/9/13/a-beginning-for-mathematics/.

OpenAI. Advisory Group on Mathematics and Artificial Intelligence, 2026. URL https://openai.com/index/advisory-group-on-mathematics-and-ai/.

Terence Tao. Mathematics in the age of AI. *arXiv preprint arXiv:2608.16753*, 2026. doi: 10.48550/arXiv.2608.16753. URL https://arxiv.org/abs/2608.16753.

William P. Thurston. On proof and progress in mathematics. *Bulletin of the American Mathematical Society*, 30(2):161–177, 1994. doi: 10.1090/S0273-0979-1994-00502-6. URL https://arxiv.org/abs/math/9404236.

Erik Y. Wang, Sumeet R. Motwani, James V. Roggeveen, Eliot Hodges, Dulhan Jayalath, Charles London, Kalyan Ramakrishnan, Jakob Foerster, Cheng Zhang, Flaviu Cipcigan, Philip Torr, and Alessandro Abate. Measuring progress in reasoning toward mathematical discovery with automatic verification. *arXiv preprint arXiv:2603.15617*, 2026. doi: 10.48550/arXiv.2603.15617. URL https://arxiv.org/abs/2603.15617.

A Evaluation of the Semantic Judge

To test whether an LLM semantic judge could distinguish legitimate functions from cheating attempts, we prepared 51 synthetic problems on Dyck paths: 17 for q-statistic discovery, 17 for t-statistic discovery, and 17 for bijection discovery. We asked GPT-6 Astra to generate two submissions for each problem: one legitimate and one adversarial, designed to hide a cheating strategy that would be difficult for the judge to detect. We also instructed the model to make the submissions similar in several respects, such as code length and coding style.

We reviewed all 102 submissions and confirmed their intended labels. We then evaluated GPT-6 Astra (Medium) as a semantic judge on this synthetic dataset. Each submission was judged separately in a fresh context containing the problem statement, submitted code, and cheating taxonomy. The judge had no access to external tools and a time limit of 120 seconds per evaluation.

The judge correctly classified all 51 legitimate submissions and all 51 cheating attempts, completing every evaluation in under 14 seconds.

Although the problems were synthetic and the functions not the most complex, these results show that Astra could quickly recognize adversarial cheating attempts in this dataset. Its accuracy supports our use of an LLM judge informed by the cheating taxonomy in the qtBench evaluation pipeline.

We illustrate the experiment below with one problem and its two submissions, one legitimate and one cheating.

A.1 Example: boundary runs of Dyck paths

Problem. Let $\mathcal{D}_n$ be the set of words with n letters N and n letters E such that every prefix contains at least as many N's as E's.[2] For $D \in \mathcal{D}_n$, let $a(D)$ be the length of its initial run of N's and $b(D)$ the length of its terminal run of E's, with both equal to zero for the empty path. Construct a uniform, combinatorially meaningful bijection $\phi : \mathcal{D}_n \to \mathcal{D}_n$ satisfying

$$a(\phi(D)) = b(D), \qquad b(\phi(D)) = a(D).$$

Supply mutually inverse Python functions `forward(path)` and `inverse(path)`, and explain the map through the structure of paths. Such a map explains the symmetry of the polynomials $F_n(q,t) = \sum_{D \in \mathcal{D}_n} q^{a(D)} t^{b(D)}$.

Legitimate submission. The first submission interprets the path as the contour of a plane rooted tree and reverses the order of children at every vertex. This is equivalent to reversing the word and interchanging N and E. The transformation preserves Dyck paths, exchanges the two boundary runs, and is an involution.

```
def decode(path):
    root = []
    stack = [root]
    for c in path:
        if c == 'N':
            child = []
            stack[-1].append(child)
            stack.append(child)
        else:
            stack.pop()
    return root

def encode(tree):
    output = []
    pending = [('visit', tree)]
    while pending:
        kind, value = pending.pop()
        if kind == 'letter':
            output.append(value)
        else:
            for child in value:
                pending.append(('letter', 'E'))
                pending.append(('visit', child))
                pending.append(('letter', 'N'))
    return ''.join(output)

def forward(path):
    return encode(decode(path))

def inverse(path):
    return encode(decode(path))
```

[2] This is the definition of Dyck paths as in Section 3.

Judge assessment (legitimate). The judge classified this submission as legitimate and gave the following explanation:

> `decode` constructs the rooted ordered tree encoded by the Dyck path. In `encode`, the LIFO stack processes each node's children in reverse order, thus reflecting the tree at every node. This uniform structural involution exchanges the leftmost and rightmost branch lengths, which are the initial N-run and terminal E-run. Both functions implement this same involution; there is no target lookup, fiber ranking, enumeration, or finite-range fitting.

Cheating submission. The second submission enumerates the paths with boundary statistics (a, b) and (b, a), then maps each path to the one with the same position in the other list. This produces a bijection because the two lists have equal size, but gives no structural explanation for that equality. It is therefore a rank-matching cheating attempt under the taxonomy, even though it satisfies the numerical identities. The judge accepted the first submission and flagged the second.

```
def boundaries(path):
    return (len(path) - len(path.lstrip('N')), len(path) - len(path.
    rstrip('E')))

def completions(n, first, last):
    entries = [(i, 'N') for i in range(first)] + [(first, 'E')]
    entries += [(2 * n - last - 1, 'N')]
    entries += [(i, 'E') for i in range(2 * n - last, 2 * n)]
    required = dict(entries)
    if any((required[i] != c for i, c in entries)):
        return []

    def extend(prefix, height):
        i = len(prefix)
        if height < 0 or height > 2 * n - i:
            return []
        if i == 2 * n:
            return [prefix]
        return [word for c, step in [('E', -1), ('N', 1)] if required.get
    (i, c) == c for word in extend(prefix + c, height + step)]
    return extend('', 0)

def forward(path):
    if not path:
        return ''
    x, y = boundaries(path)
    n = len(path) // 2
    source = completions(n, x, y)
    target = completions(n, y, x)
    return target[source.index(path)]

def inverse(path):
    return forward(path)
```

Judge assessment (cheating). The judge classified this submission as cheating and gave the following explanation:

> `completions` exhaustively enumerates the Dyck paths with specified boundary runs in a fixed E-before-N order. `forward` builds the (x, y) and (y, x) fibers and returns `target[source.index(path)]`, pairing paths solely by their enumeration ranks rather than transforming path structure. This is a uniform rank-matched bijection implemented by exhaustive fiber enumeration, not a combinatorially meaningful construction.

B Overview of Problems and OpenAI's Proposed Solutions

Table 2 lists all 30 problems of qtBench with the status of the corresponding proposal in the OpenAI package. The package contains 17 proposed constructions and 3 proposed counterexamples. It includes code for 10 of the constructions and scripts for the 3 counterexamples; the other 7 constructions are write-ups only.

The marks are our informal assessment, they do not certify correctness or novelty, and they are not benchmark scores. In particular, we assessed every proposal in the package, including any that the checker or the LLM judge rejected; the seven constructions without code could not be run through the checker at all. A tick (✓) marks a direct construction, which computes the statistic or bijection from the structure of each object, as asked in Section 3, and the construction passes both the checker and the LLM judge assessment. A question mark (**?**) marks proposed solutions that we are still assessing. These constructions are not implemented in code (except for problem 28), hence the checker did not validate or reject them. A cross (✗) marks a construction that is clear cheating and of no mathematical value; such construction may pass the checker but not the LLM judge assessment.

ID	Problem	Type	Code	Assessment
1	Noncrossing q,t-Narayana partner (calibration)	t		—
2	Dyck paths: area/bounce exchange	B	no	**?**
3	Type B q,t-Catalan area partner	t		—
4	Polyominoes: area/bounce exchange	B	no	**?**
5	Polyominoes: (m,n)-transpose bijection	B	yes	✓
6	Labelled polyominoes: Theta-operator area partner	t	yes	✓
7	Doubly decorated Dyck paths: unified-Delta partner	t	no	✗
8	Shareshian-Wachs e-positivity statistic	q		—
9	Zero-rooted tiered trees: Ξ-operator partner	t	yes	**C**
10	Multi-labelled Dyck paths: super-nabla partner	t	yes	**C**
11	Rise-decorated rectangular paths: Delta partner	t	yes	**C**
12	Tamari Hilbert-series shadow: third statistic	t		—
13	Modified (q,t)-Kostka tableau statistics	t		—
14	Unicellular LLT Schur-coefficient statistic	q	no	✗
15	Involutions: orbit-harmonics Hilbert-series statistic	q	yes	✓
16	Matchings: Jack connection-coefficient statistic	q		—
17	Permutations: q-Eulerian gamma statistic	q	yes	✓
18	Selected-area gamma-parking: Delta-Ξ partner	t		—
19	Rooted tiered trees: Theta-operator partner	t		—
20	Selected-area lattice gamma-parking Schur partner	t		—
21	Threshold-graph trees: flow-polytope Ehrhart partner	t		—
22	Promotion tableaux: intrinsic statistic	q	yes	✓
23	Noncrossing partitions: q-Kreweras statistic	q	yes	✓
24	Macdonald fillings: $\mathrm{inv}/\mathrm{maj}$ exchange	B	no	**?**
25	Parking functions: area/dinv exchange	B	no	**?**
26	Connected graphs: sibling/tuft exchange	B	yes	✗
27	Alternating sign matrices: DPP-weight statistic	q	no	**?**
28	Shifted set-valued tableaux: P/Q correspondence	B	yes	**?**
29	Andrews-Bressoud successive-rank bijection	B	yes	✗
30	Partition matrices: inversion-sequence bijection	B	yes	✓

Table 2: Status of the OpenAI proposals for the 30 qtBench problems. *Type*: q-statistic (q), t-statistic (t) or bijection (B) discovery, as in Table 1. *Code*: whether the package includes code (for counterexamples, a verification script). The marks are an informal assessment.

Counterexamples. The three counterexamples concern the target polynomials, not candidate statistics. Each exhibits a negative coefficient outside the public data range: for Problem 9, the coefficient of q^1t^0 at tier sizes $(10,9,9)$ is $-2{,}871{,}205{,}236{,}900$; for Problem 10, the constant term at $n=7$, $k=2$ is $-8{,}484$; and for Problem 11, the constant term at $(m,n,k)=(2,11,1)$ is -528. The generating function of a statistic has nonnegative coefficients, so no statistic can generate these polynomials. These counterexamples are of considerable value to algebraic combinatorics community.