Using Large Language Models to Generate Claims and Counterexamples in Graph Theory

Chase Carstensen
Worcester Polytechnic Institute
Worcester, MA
wcarstensen@wpi.edu

Jugal Kalita
University of Colorado Colorado Springs
Colorado Springs, CO
jkalita@uccs.edu

Abstract

Automated mathematical reasoning has been a popular research topic in natural language processing (NLP) recently. Recent results are focused on constructive tasks like theorem proving, with less attention on other tasks such as counterexample generation, especially using large language models (LLMs). This task is overlooked, as counterexample generation plays a crucial role in many mathematical reasoning tasks. In this work, we construct and analyze two datasets, one of simple, and another of challenging, false graph-theoretic claims to evaluate the capabilities of LLMs on this task. A large proprietary model (GPT-4.1) falsified 85% of claims in the simple set, and 39.5% of claims in the challenge set. On the other hand, a leading open-source model (Qwen Math 2.5 7B) falsified less than 5% of claims in the simple set. We find that while LLMs excel at generating false claims, the primary bottleneck in creating such synthetic benchmarks at scale is claim verification. Guided search methods, including the deep cross-entropy method and evolutionary algorithms, proved computationally infeasible for disproving these claims. This work establishes a baseline for LLM-based counterexample generation in graph theory, highlighting that the verification bottleneck and significant disparities in reasoning capabilities between models are key challenges for the field.

1 Introduction

Mathematicians have long been making use of computational tools and artificial intelligence (AI) in order to solve problems. In 1956 Newell and Simon used their "logic theory machine" to prove a number of theorems from *Principia Mathematica*, and this was one of the first times computers were able to perform tasks, such as mathematical reasoning, that were seen as uniquely human (Newell and Simon, 1956). These rule-based approaches were followed by proof assistants, exhaustive searches, and guided searches (Appel, 1984; de Bruijn, 1983; Bundy, 1988).

Throughout the history of AI-for-mathematics, much of the effort has been spent on constructive ideas (such as finding solutions to math word problems and writing proofs) rather than falsifying claims by finding counterexamples (Lu et al., 2023).

Even today, searching for counterexamples remains a small niche, with recent landmark work in mathematical reasoning being improvements to automated theorem provers and the math problem-solving abilities of LLMs (Ren et al., 2025; Yang et al., 2024).

There are questions as to if language models actually understand the underlying mathematical structures when problem solving (Mirzadeh et al., 2024). The task of finding counterexamples offers a possible way to test this distinction, as studies in mathematics pedagogy suggest that counterexample generation is an important part of human mathematical reasoning and understanding (Komatsu, 2010). To date there has been little explicit experimentation on the generation of mathematical counterexamples using LLMs, especially in the domain of graph theory.

To illustrate the task of counterexample generation, consider the simple, false claim that "All connected graphs contain a Hamilton cycle". A graph is connected if there is a path between any two of its vertices, and a Hamilton cycle is a cycle that visits every vertex exactly once. This claim can easily be falsified by a counterexample. For instance, the path graph on four vertices (P_4) is connected but has no cycles, so it cannot contain a Hamilton cycle. The core of this research is to assess an LLM's ability to understand the claim and generate a valid counterexample like P_4.

With this in mind, we seek to answer the following research questions.

- **RQ1:** How can we generate and verify false graph theoretic claims at scale?
- **RQ2:** To what extent can current LLMs generate valid counterexamples for such false claims in graph theory?

This work attempts to answer these questions with 3 main contributions.

1. Developed two datasets of simple and challenging false graph theoretic claims for use in NLP-for-mathematics research.
2. Benchmarked the ability of state-of-the-art (SOTA) open- and closed-source LLMs to find counterexamples to claims in the datasets.
3. Conducted a comparative analysis of claim generation techniques and identified key bottlenecks to establish the current state of the art.

2 Related Work

2.1 Automated Mathematical Reasoning

Historic approaches to automated mathematical reasoning were based on symbolic methods. Symbolic reasoning involves encoding mathematical problems in formal language and using explicit rules and logic to solve the problem. Examples of symbolic methods include proof assistants (De Moura et al., 2015), SMT solvers (De Moura and Bjørner, 2008), first-order provers (Riazanov and Voronkov, 2001), and computer-algebra systems (pau, 1996).

As machine learning matured as field, neural methods came to dominate automated mathematical reasoning (Wang et al., 2017). These methods learn vector representations of mathematical statements to detect patterns in a large amount of data (Saxton et al., 2018).

Most recently, there has been a rise in neuro-symbolic approaches. Neuro-symbolic methods combine the previous two reasoning paradigms. Often this involves using neural methods to generate some construction which can be verified by a separate symbolic engine. Interest in neuro-symbolic methods has been increasing recently, showing promising results in the field with systems such as GPT-f and LeanDojo (Polu and Sutskever, 2020; Yang et al., 2023).

2.2 Automated Counterexample Generation

Computers have long aided in the search for counterexamples in mathematics, and advancements in machine learning techniques have heavily impacted the field. Reinforcement learning and transformer-based models have already been successfully found constructions to disprove long-standing open conjectures (Wagner, 2021; Charton et al., 2024).

2.2.1 LLM-based Counterexample Generation

A relatively small amount of prior work has assessed an LLM's ability to disprove mathematical claims. Previous work shows that current LLMs can struggle to find and use counterexamples while reasoning (Li et al., 2025). Sinha et al. (2025) demonstrates that LLMs were able to independently solve programming challenges more often then they could find counterexamples to rule out incorrect solutions to the same challenges.

2.3 Benchmark Datasets for Mathematical Reasoning

There are many popular, NL-focused benchmarks and datasets for evaluating mathematical reasoning in LLMs, such as MATH, GSM8k, and more recently FrontierMath (Hendrycks et al., 2021; Cobbe et al., 2021; Glazer et al., 2024). Each of these datasets include natural language problem statements and solutions. Because of problem-solving nature of the dataset rather than claim verification, these datasets are not well suited for the task of counterexample generation.

Counterexample focused datasets are scarce, especially so for claims in graph theory specifically. One of the only truly counterexample based datasets, CounterMath, is not public and does not contain any problems in graph theory (Li et al., 2025).

2.4 Reasoning in LLMs

Despite exhibiting human-level performance on many tasks, many LLMs trained on general knowledge may struggle generate multi-step mathematical reasoning (Cobbe et al., 2021). In light of this, there are a number of distinct strategies for eliciting mathematical reasoning from LLMs.

Chain-of-though (CoT) prompting involves providing annotated examples to an LLM which describe the sequence of steps used to come to a solution to a given problem, or explicitly instructing a model to reason through a task step by step (Wei

et al., 2022). CoT has been shown to improve performance on many mathematical reasoning tasks.

In addition to CoT, LLMs are able to critique their own responses and correct inaccuracies. Madaan et al. proposed a system that iteratively generates solutions to tasks, critiques them, and then refines them until a stop condition is met (Madaan et al., 2023). Their algorithm showed performance improvements across a variety of domains including mathematical reasoning.

Given the proven effectiveness of these methods, we incorporate both chain-of-thought and self-refinement prompting in our benchmarking experiments to ensure a robust evaluation of each model's capabilities.

3 Problem Statement

3.1 Formal Definitions

Let $\mathcal{X}$ be the set of all constructions (most likely graphs in this project) and let the set of all mathematical statements

$$\mathcal{S} = \{s \mid s : \mathcal{X} \to \{\text{true, false}\}\}$$

be given, and let the set of all conditions

$$\mathcal{C} = \{c \mid c : \mathcal{X} \to \{\text{true, false}\}\}$$

be given.

A claim P is a pair of a statement $s \in \mathcal{S}$ and a set of conditions $C \subseteq \mathcal{C}$. $P = (s, C)$ is true if and only if

$$s(x) = \text{true } \forall x \in \mathcal{X} \text{ such that } c(x) = \text{true } \forall c \in C$$

Less formally, a claim is true if the statement is true for all x such that all conditions are met. The set of all claims is defined by $\mathcal{P} = \mathcal{S} \times \mathcal{P}(\mathcal{C})$.

Here, given a construction $x \in \mathcal{X}$ and a claim $P \in \mathcal{P}$, then x is a counterexample of $P = (s, C)$ if and only if

$$(c(x) = \text{true } \forall c \in C) \wedge (s(x) = \text{false})$$

In other words, a construction is a counterexample if it meets all conditions but fails to make the statement true.

In this project we attempt to evaluate an LLM-based system $f : \mathcal{P} \to \mathcal{X} \cup \{\perp\}$ where given a claim P in natural language we have

$$f(P) = \begin{cases} x & x \text{ is a counterexample of } P \\ \perp & P \text{ is true} \end{cases}$$

3.1.1 Example Claim

Let a claim $P^* = (s, C)$ be given in natural language as "All connected graphs contain a Hamilton cycle."

In our formal framework

- $\mathcal{X}$ is the set of all graphs, therefore $x \in \mathcal{X}$ is a single graph
- $C = \{c\}$ where $c(x) =$ "x is a connected graph."
- $s(x) =$ "x contains a Hamilton cycle."

3.2 Claim Generation and Benchmarking

The primary objectives of this work are to systematically generate a corpus of both simple and complex, false graph-theoretic claims and evaluate how well LLMs can falsify these claims. The claim collection process is broken down into two major steps, claim generation, where a natural language graph-theoretic claim is produced, and claim verification, where a counterexample is found to prove the falsity of a claim.

We aim to measure the accuracy of a variety of LLMs in constructing counterexamples to the false claims generated, and also determine how an LLM fails when it does not succeed in generating a counterexample. This will help understand the graph reasoning ability of LLMs and indicate if these models are actually reasoning or just mimicking human behavior.

4 Methodology

4.1 Claim Generation

Throughout this work, three main strategies for claim generation were tested, extracting false claims and altering theorems from the literature, generating claims programmatically, and generating claims using a LLM.

4.1.1 Claim Extraction

Early strategies for generating a dataset of false graph theoretic claims, depicted in Figure 4, were based on automatically scraping results (such as explicit counterexamples and theorems) from recent papers using LLMs. With these extracted results, false claims and counterexamples were stored, and an LLM was provided with proven statements from a paper to generate plausible-sounding but false variations. For instance, by swapping a condition or relaxing a conclusion.

While we initially explored this approach, it proved to be difficult to scale due to a small amount of recent results and issues verifying extraction accuracy, so we switched our approach and looked at more flexible methods.

4.1.2 Programmatic Generation

Our programmatic approach to claim generation involved collecting lists of graph properties, and randomly sampling these lists of properties to create a set of conditions and a statement. A basic template was used to create the natural language claims from the raw conditions and statement. These property lists were populated with simple, undergraduate-level graph-theoretic topics such as connectivity, coloring, cycles, and matchings. Ideally this strategy would create claims which were easily verifiable as false, and provide us with a set of simple claims to initially benchmark LLM performance.

In addition to a random sampling approach, we also implemented a simple dependency graph to endow the system with some basic graph theoretic knowledge. During the claim generation process, the system would perform a breadth first search through the dependency graph and eliminate incompatible properties from the sampling space.

For instance, using a purely random sampling approach, it would be entirely possible for the system to generate a claim such as, "If G is a tree and G is acyclic, then G is Hamiltonian." This claim has redundant conditions, because if G is a tree, then it is implied that G is acyclic. Additionally, since G is acyclic, it certainly cannot contain a Hamilton cycle. The dependency graph would prevent statements with these types of basic redundancies and contradictions from being generated.

However, we found this intelligent generation strategy greatly increased system complexity for only small gains in claim quality. Many subtle graph-theoretic relationships were difficult to encode, limiting the dependency graph's effectiveness. Given this trade-off, our final simple dataset was generated using the random sampling approach to ensure scalability.

4.1.3 LLM-Based Generation

Programmatically generated claims were simple, being restricted to a relatively small set of graph-theoretic properties and statements. In order to provide more difficult benchmarking material for models which perform well on the simple testing set, we employed LLMs in the claim generation process as well. To achieve this, the LLM was instructed to generate a plausible sounding yet false claim based on one of nine graph-theoretic themes. These themes include similar topics to the programmatic approach (connectivity, coloring, cycles, and matchings) along with more complex ideas like spectral graph theory, graph symmetries and automorphisms, extremal graph theory, and graph decompositions.

4.2 Claim Verification

To generate a dataset of false claims, we clearly need a way to ensure that the claims we collected are actually false. To do this we used NetworkX, a Python library with robust graph tooling (Hagberg et al., 2008). Using NetworkX and NumPy, we developed predicates for all of the conditions and the statement of every claim.

For programmatic generation, given the relatively small size of the sample lists, these NetworkX predicates were generated by hand. For LLM-based claim generation, in order to avoid restricting the LLM to a small set of hand made predicates, the NetworkX predicates for these claims were also generated by the LLM. The LLM predicate generation process included a simple refinement loop. Each generated predicate was tested on a small number of random graphs, and if any exceptions were thrown, the LLM was instructed to fix the specific bug. This phase caught a vast majority of simple runtime errors in the LLM-generated code, and a more thorough manual debugging phase was conducted after all code was generated in order to catch residual runtime errors.

With a way to determine if an individual graph is or is not a counterexample to a given claim, we needed ways to generate candidate graphs.

4.2.1 Exhaustive Search

Given the simple nature of the programmatically generated claims, the easiest search strategy to implement was an exhaustive search. All non-isomorphic graphs on a small number of vertices were enumerated and checked against the necessary predicates. This allowed us to quickly find counterexamples to claims with small minimum counterexamples. The obvious drawback is that this is not at all scalable, and realistically only graphs on a single digit number of vertices can be checked due to how the search space grows combinatorially.

Some optimizations were made to this system, including the use of optimized graph tools such as

geng from "nauty and Traces" to quickly generate many non-isomorphic graphs which meet certain conditions (McKay and Piperno, 2014). These optimizations did little to help the scalability of this strategy as the search space becomes too large very quickly.

Despite this, the pairing of programmatic claim generation and exhaustive counterexample search was rather effective at collecting a large number of claims for the first testing set. That being said, the LLM-generated claims were far too complex for as simple of a search strategy such as an exhaustive search.

4.2.2 Guided Searches

In order to disprove the complex LLM-generated claims we had to implement a more creative search strategy. Many modern search strategies require a scoring function, or a function which quantifies "how close" a candidate construction is to being a counterexample to a claim. Since these functions are complex and claim specific, we once again utilized LLMs to accelerate development. The scoring functions went through the same automated and manual debugging processes that were applied to the NetworkX predicates.

Once these scoring functions had been generated and debugged, we implemented two main guided counterexample searches. First, we implemented Wagners's deep cross-entropy method (DCEM) (Wagner, 2021). This is a reinforcement learning based search which uses a neural network to approximate a graph construction policy by using it to auto-regressively build a graph edge by edge. Algorithm 1 shows a high level overview of how this method was implemented. It is worth noting that in Wagner's approach, he used one-hot positional encodings to inform the network of which edge it was currently acting on. The number of edges in a simple, undirected graph on n vertices is $\frac{n(n-1)}{2}$, and thus the positional encoding would have size $\frac{n(n-1)}{2}$. This does not scale well, as it grows quadratically with respect to n, so in our implementation we used the fixed-length sinusoidal positional encodings made popular by the development of the transformer architecture (Vaswani et al., 2017).

In addition to the DCEM, we also implemented an evolutionary algorithm (EA)-based counterexample search (Algorithm 2). This search involved initializing a random population of graphs, determining the fitness of individual graphs using the

Algorithm 1 The Deep Cross-Entropy Method

$n \leftarrow$ size of graph constructions
$N \leftarrow$ number of constructions
$M \leftarrow$ initialize a neural network
$C \leftarrow$ initialize an empty list
$k_1 \leftarrow$ % of top constructions to train M on
$k_2 \leftarrow$ % of top constructions to survive each generation
while top construction not a counterexample **do**
  for _ from 1 to N **do**
    $w \leftarrow$ initialize a list of size $\frac{n(n-1)}{2}$
    for i from 1 to $\frac{n(n-1)}{2}$ **do**
      $pe \leftarrow$ positional encoding
      $a \leftarrow M(w \| pe)$ ▷ Action from M
      $w[i] \leftarrow a$
    end for
    $C \leftarrow C + w$
  end for
  $S \leftarrow$ score each construction
  $C \leftarrow$ sorted based on S
  $T \leftarrow$ top $k_1\%$ of C
  for each construction c in T **do**
    $M \leftarrow$ update weights of M based on c
  end for
  $C \leftarrow$ top $k_2\%$ of C
end while

LLM-generated scoring functions, then through uniform crossover among elite individuals and random mutation, new offspring were generated. After each generation, a portion of the least fit graphs were culled to keep the population size constant.

Algorithm 2 Evolutionary Counterexample Search

```
n ← size of graph constructions
N ← population size
c ← % of children to generate
m ← initial mutation rate
P ← initialize a population of N random graphs
while top construction not a counterexample do
    C ← initialize an empty list
    E ← top k% of P
    for _ from 1 to ⌊N · c⌋ do
        p1, p2 ← two random graphs from E
        w ← uniform crossover of p1 and p2
        w ← random mutation with rate m
        C ← C + w
    end for
    P ← P + C
    P ← top N individuals in P
    if top score in P is stagnating then
        increment m slightly
    else
        reset m
    end if
    if top score in P severely stagnates then
        reinitialize P, keeping best individuals
        reset m
    end if
end while
```

4.3 LLM Benchmarking

The pipeline used for benchmarking LLMs uses CoT prompting and a basic refinement loop (seen in Figure 5). The LLM is instructed to provide the counterexample graph as an adjacency matrix, which is then parsed and verified using the premade NetworkX predicates. If the LLM successfully generated an adjacency matrix, but the graph it represents is not a counterexample, then the LLM will be informed of which part of the claim was violated and instructed to alter the graph slightly.

5 Results

5.1 Claim Generation

Methods for extracting claims were unsuccessful. Papers were sourced from arXiv using keyword searches such as "refute," "counterexample," and "graph theory." We then used OpenAI's GPT o4-mini to extract the desired results from each of the papers sourced.

The LLM did not reliably extract information from these dense and technical papers. For example, in his paper Wagner used the DCEM to find a counterexample to a conjecture in spectral graph theory (Wagner, 2021). This graph has 203 vertices in total, but in the text he discusses how this graph is P_{13} with 190 pendant vertices added to a single vertex. When given this paper, o4-mini was slightly confused and said that this counterexample had 190 vertices total, missing the fact these 190 vertices were added to P_{13}. Additionally, yields were very low. After scraping 15 papers, only 25 false claims had been sourced. A very large number of papers would need to be analyzed to generate sufficient benchmarking material, and there simply are not that many recent papers in the field. This led to this approach being abandoned early on.

Following claim extraction, we moved to generative approaches. Random generation was immediately much more scalable than extraction methods, as it could generate thousands of claims in seconds. To evaluate this claim generation approach, we generated 2000 claims, and analyzed the output. We discovered that although they were intended to be simple, $\geq$30.95% of the 2000 claims had direct contradictions and an additional $\geq$6.95% had redundant properties. It is not possible to get exact totals of redundant or contradictory properties, so these values serve as lower bounds.

For example, this system generated the claim "Let G be a claw-free graph on $2 \leq n \leq 4$ vertices such that $|E(G)| = 3$ and $\delta(G) \geq 1$. Then G contains a perfect matching." This may read like a reasonable claim, but there are some obvious issues. The most significant issue is that this claim allows for odd vertex counts, but in order for a graph to have a perfect matching, a graph must have an even number of vertices. A graph on 3 vertices cannot possibly satisfy the statement.

As previously mentioned, we implemented a dependency graph to prevent these basic contradictions and redundancies, but complex graph-theoretic properties were difficult to encode using this basic approach, which limited the effectiveness of the dependency graph. We once again generated 2000 claims with this approach and these claims were marginally better, with only $\geq$16.9% of claims having contradictions and $\geq$0.85% hav-

ing redundant properties. This solution greatly increased the complexity of the system, and harmed the scalability. Adding new properties to the system involved manually updating the entire dependency graph and the generation of each claim would preform a breadth first search across the dependency graph which added significant overhead.

The LLM-generated claims were much higher quality, and rarely contained the same type of errors present in the programmatically generated claims. We generated 450 claims using this approach, across nine different subfields of graph theory (50 claims each). The main tradeoff here is between increased claim quality and the lack of an assurance that the necessary NetworkX predicates and scoring functions are implemented correctly. 9.33% of LLM-generated predicates contained uncaught runtime errors and required manual intervention.

An example of an LLM-generated claim is "Every cubic 3-edge-connected graph can have its edge set decomposed into two spanning trees." This claim is much more plausible sounding compared to the previous matching claim. It does not contain any mutually exclusive conditions, and the issue with the claim is just in the quantifier "Every." This statement does hold for some graphs and not others, which makes it subtly false. The LLM-generated claims also span a greater number of themes and have much greater variety than those generated programmatically.

Once these claims had been generated, they needed to be verified as false.

5.2 Claim Verification

The exhaustive search we implemented proved to be useful in determining what claims had a small minimum counterexample. When constrained to small graphs, the system could generate and verify all non-isomorphic graphs using the necessary predicates within just a few minutes at most. The search was practical for finding counterexamples on up to $n = 8$ vertices, beyond this the combinatorial explosion of non-isomorphic graphs made runtimes prohibitively long.

This method was effective in combination with our programmatic generation methods. It allowed us to create our simple dataset generation pipeline as shown in Figure 6. Using this, we generated 4017 false graph-theoretic claims in a matter of hours, as the task is easily parallelizable across multiple CPU cores. These claims had an average minimum counterexample size of 4.283 vertices. Claims for which no counterexample was found and claims for which all graphs are counterexamples were discarded in order to include only false and non-trivial claims in the simple dataset. Figure 7, Figure 8 and Figure 9 show the distributions of statements, conditions and types within the dataset. In these graphs, an interesting trend appears where the most restrictive statements occur more often, and the most restrictive types and conditions occur less often. This is likely because more restrictive types and conditions greatly limit the counterexample search space, whereas restrictive statements grow it.

The guided searches that were implemented faced some fundamental challenges that greatly limited its effectiveness. While exhaustive search was sufficient for our simple dataset, the more complex, LLM-generated claims often have minimum counterexamples with more vertices, placing them beyond the reach of brute-force methods.

The primary challenge for these guided searches was computational cost. Both methods require many thousands of iterations to navigate the vast search space, with each iteration involving expensive operations like neural network inference or population-wide fitness evaluations.

To quantify this issues, we benchmarked both DCEM and the EA on three LLM-generated claims.

- **Claim 52:** Every cubic 3-edge-connected graph contains a Hamiltonian cycle.
- **Claim 124:** Every triangle-free graph with chromatic number 4 has exactly one vertex of maximum degree.
- **Claim 333:** For any connected d-regular graph G of order n and girth at least 5, let $\delta = d - \max_{i \neq 1} \left|\lambda_i(A(G))\right|$ be the adjacency spectral gap, and let $t(G)$ be the number of triangles in G. Then $t(G) \geq \frac{n\,\delta^3}{6d^2}$.

Each search algorithm was run with a fixed time budget of 60 minutes per claim.

The progression of the searches throughout the hour can be seen in Figure 1 and Figure 2. It shows how for almost all of the searches, the fitness scores of the best candidate graphs typically improved during the initial iterations, they almost always plateaued at a low score, indicating the search had become stuck at a local optimum. In the EA-based

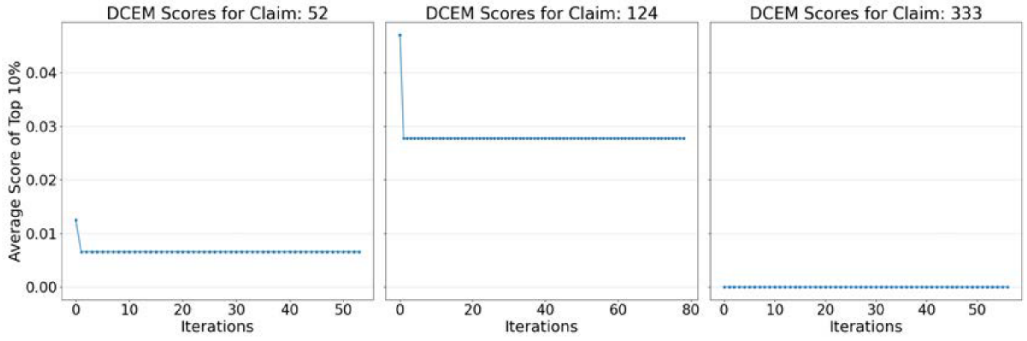

Figure 1: Benchmarking Results for the DCEM

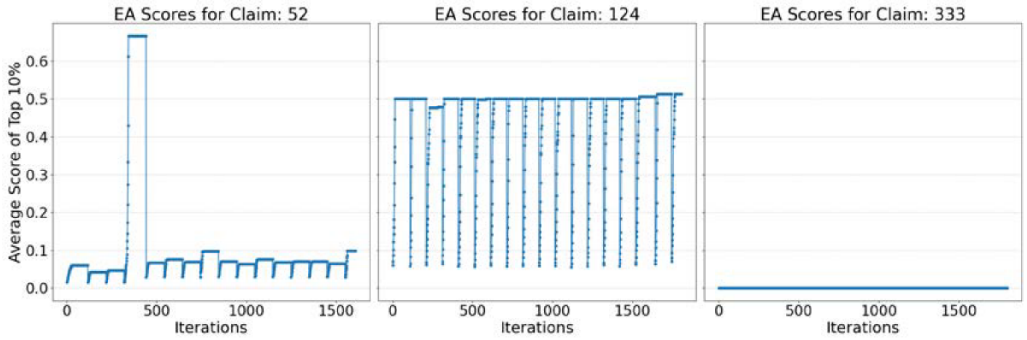

Figure 2: Benchmarking Results for the EA

search for claim 52, early on a highly fit individual was found and its traits spread across the population. After the scores stagnated and a large portion of the population was re-initialized, this construction was unable to dominate and pass its traits to offspring. This suggests that for this claim there is a subtle yet substantial gap between low and high scoring graphs.

The results confirmed that these methods are not practical for large-scale verification. Because of this, the LLM-generated claims remained unverified. In all trials, neither algorithm found a counterexample within the time limit. This experiment demonstrates that verifying complex, plausible claims is the critical bottleneck in our pipeline. While LLMs can generate such claims with ease, the tools to efficiently prove their falsehood are not yet mature enough for this task, making it a key area for future work.

5.3 LLM Benchmarking

Despite not being able to formally disprove the LLM-generated claims, we used both datasets to benchmark LLM performance. We evaluated OpenAI's GPT-4.1 and Alibaba's open-source Qwen Math 2.5 7B. For each claim, models were scored based on whether they produced a valid counterexample. We categorized failures into three types, generating an incorrect graph (a graph that met the conditions but did not falsify the statement, or vice versa), incorrectly determining that the claim was true, or producing a format error (e.g., an invalid adjacency matrix).

The results on the simple, programmatically generated dataset are summarized in Table 1. The leading proprietary model, GPT-4.1, demonstrated strong performance, correctly falsifying 85% of the claims. In contrast, the open-source models struggled significantly. Qwen Math 2.5 7B, despite being specialized for mathematical tasks, only succeeded on 4.7% of claims, frequently incorrectly claiming that the claim is true. This wide performance gap highlights the current disparity between closed- and open-source models on this type of structured reasoning task.

On the more challenging LLM-generated dataset, the performance of all models dropped significantly, as shown in Table 2. Even though these claims had not yet been proven false, we could still use the pre-made NetworkX predicates to verify the LLM's constructions.

Even GPT-4.1 only managed to find counterex-

Table 1: LLM Performance on the Simple Dataset

Model	Accuracy	Incorrect Graph	Claimed True	Format Error
GPT-4.1	85%	2%	14%	0%
Qwen Math 2.5 7B	4.7%	0%	79.3%	15.9%

Table 2: LLM Performance on the Challenging Dataset

Model	Accuracy	Incorrect Graph	Claimed True	Format Error
GPT-4.1	39.5%	23.3%	15.3%	21.7%
Qwen Math 2.5 7B	0.8%	6.6%	33.5%	58.8%

amples to 39.5% of the claims. The rate at which GPT incorrectly asserted a claim was true also increased substantially. GPT-4.1 struggled in particular with questions relating to graph decompositions and graph symmetries and automorphisms. Figure 3 shows that GPT-4.1 found significantly fewer counterexamples for these claims, 16% and 20% respectively, as compared to other topics which averaged 45% accuracy.

These results confirm the difficulty of the LLM-generated claims and expose the counterexample reasoning abilities of current LLMs.

6 Analysis

These results can tell us a lot about the verification bottleneck that this problem faces, the two tiers of LLM performance on counterexample reasoning tasks in graph theory, and the overall implications to AI for mathematics.

6.1 Verification Bottleneck

The two tasks that claim generation was split into are very asymmetric. LLMs can easily conjecture plausible sounding graph theoretic conjectures, but the system faces massive computational difficulty when attempting to search for counterexamples.

The two guided searches we implemented, failed to efficiently find counterexamples to these claims at scale. This is likely due to two main factors. One being the incredibly complex landscape the this search needed to traverse. The plausible nature of the claims provides a complicated search space with many graphs which may pass as being "close" to being a counterexample, but may subtly violate a condition, creating a local optimum in the search space. As Figure 2 shows, it was common for the searches to get stuck at a local optimum, and it was difficult for them to break out of these extrema.

Another factor is the LLM-generated scoring function. Despite over 50,000 graphs being generated and tested, none received a score above zero for claim 333. The LLM-generated scoring function immediately provides disconnected or non-k-regular graphs a score of zero, rather than any sort of measure as to how close they are to being connected or k-regular. Since the likelihood of either search algorithm essentially randomly generating a connected and k-regular graph with no signal from the scoring function is incredibly low, this led to both searches being completely stuck on this claim. Generating high quality scoring functions using LLMs for complex graph-theoretic claims is difficult and is itself an interesting potential research direction.

A further complication was hyperparameter tuning. In line with the no free lunch theorems for optimization, we found that a set of hyperparameters (like learning rates or population sizes) that worked for one claim often performed poorly on another, as each claim represents a unique optimization landscape (Wolpert and Macready, 1997). Manually tuning these parameters for each individual claim is not a scalable approach for automated dataset creation.

Despite the issues our specialized searches faced, LLMs like GPT-4.1 were still able to find a significant portion of counterexamples in the challenging set, which we could not verify with our automated searches. This leads to the question, what about LLM reasoning allows them to find these counterexamples more efficiently than the specialized searches?

Examining the model outputs, during its reasoning it does appear to be using complex graph knowledge to construct counterexamples. For example, for the claim "If a simple graph G is 4-edge-connected, then G is 3-vertex-connected," GPT-4.1 was able to come up with a simple construction containing two connected K_5 graphs with shared vertices which falsified the statement. The LLM out-

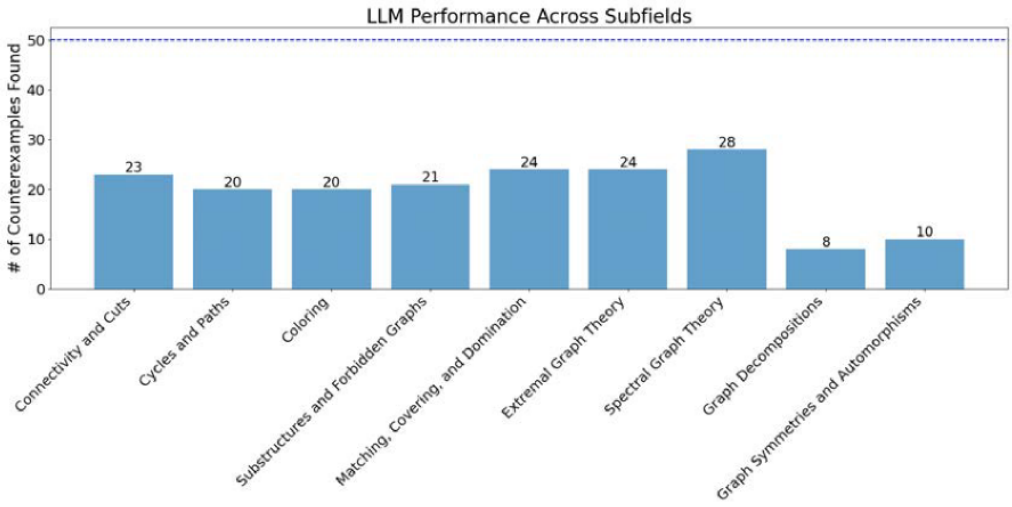

Figure 3: LLM Performance Across Graph Theory Subfields

put contained "**Idea:** Take two K_5's and connect them at just two vertices (identify, say, vertices 0 and 1)—but that gives vertex-connectivity 2, but the edge-connectivity will be more than 2 if we arrange for every cut to require at least 4 edges." This shows that the LLM has significant graph-theoretic knowledge which can guide its counterexample search. In contrast, guided searches like DCEM and EA perform a knowledge-agnostic, black-box optimization. They are entirely reliant on the scalar signal from the scoring function, which provides limited information compared to the nuanced, structural understanding the LLM appears to use.

With this in mind, it still does not explain the performance chasm between open- and closed-source models.

6.2 Two Tiers of Counterexample Construction

Our results reveal a distinct performance gap in complex, graph-theoretic reasoning, with closed-source models operating in a tier above their open-source counterparts. It is well known that large models tend to outperform smaller models on most mathematical reasoning tasks, but the difference in performance between Qwen Math 2.5 and GPT-4.1 is dramatic.

As we previously identified, GPT-4.1 was able to use its knowledge of graph theory to construct a number of unique counterexamples. On the other hand, when examining the reasoning process of Qwen 2.5 Math it was discovered that for all but one of the challenging counterexamples found by Qwen, it did not come up with a unique construction. Rather Qwen reiterated some basic graph-theoretic results and produced the adjacency matrix of a well known graph, such as a cycle graph and the Petersen graph. Qwen also had a tendency to claim that false statements were true. Qwen claimed that no counterexample existed for nearly 80% of the simple claims it was provided.

When "proving" that these claims were true and that no counterexample existed, Qwen often relied on incorrect statements of well known theorems in graph theory. When trying to find a counterexample to the claim, "Every connected triangle-free graph G with maximum degree $\Delta(G) \geq 4$ satisfies $\chi(G) \leq \Delta(G) - 1$," Qwen based its argument on an incorrect statement of Brooks' Theorem.

Qwen stated "However, it is known from graph theory that the chromatic number of a triangle-free graph G with maximum degree $\Delta(G) \geq 4$ is at most $\Delta(G) - 1$. This result is a well-known theorem in graph theory, and it has been proven by mathematicians." This is not true, Brooks' theorem states that for a graph which is not complete or an odd cycle, then $\chi(G) \leq \Delta(G)$. A triangle free graph on n vertices cannot be complete if $n \geq 3$ and since $\Delta(G) > 2$, then G cannot be an odd cycle. Therefore, the inequality Qwen stated $\chi(G) \leq \Delta(G) - 1$ does not hold for all graphs, which is important to note because many graphs, such as k-regular graphs satisfy Brooks' Theorem with equality.

Ultimately, GPT-4.1 demonstrates capabilities indicative of real graph-theoretic reasoning, allow-

ing it to construct novel objects. In contrast, smaller open-source models like Qwen 2.5 Math 7B appear to rely on retrieving and regurgitating facts, which is not only less powerful but also susceptible to factual errors making it unreliable for this task.

6.3 Implications to AI for Mathematics

One of the most promising near-term roles for LLMs in mathematics research is their conjecturing ability. It has been shown before that LLMs could generate plausible conjectures and prune obviously false claims by generating counterexamples, but this work showed that LLMs can conjecture across a variety of thematic areas across graph theory (Chuharski et al., 2024).

The automated conjecturing and pruning pipeline is currently severely limited by issue of claim verification. Automatically falsifying or proving statements remains a complex issue in mathematics, and the success of automated conjecturing systems will be limited until significant advancements are made to these techniques.

This work also showed that the while the counterexample generation ability of frontier closed-source models was poor, when they did succeed they showed some genuine reasoning abilities. This may indicate that further advancements to the mathematical reasoning abilities of LLMs, could lead to breakthroughs in the area of automated conjecturing.

7 Conclusion

In this work, we investigated the capabilities of large language models on the task of finding counterexamples to false claims in graph theory. We developed two distinct datasets, one featuring simple, programmatically generated claims and another containing more complex and plausible conjectures generated by an LLM.

Our experiments reveal a significant performance gap between models. On our simpler dataset, a large proprietary model like GPT-4.1 demonstrated strong proficiency, successfully identifying counterexamples for 85% of simple claims, and just less than 40% of challenging ones. In contrast, a leading open-source model struggled, achieving less than 5% accuracy. Performance for all models dropped substantially on the more challenging dataset, confirming that generating counterexamples for plausible, nuanced claims remains a difficult task for the current generation of LLMs.

Through this process, we addressed our primary research questions. We found that while LLMs are effective at generating high-quality, complex claims, the primary bottleneck for creating such datasets at scale is claim verification. Computationally intensive search methods like the DCEM and EAs proved infeasible for the scalable verification required to build a large-scale benchmark of difficult problems. Our work establishes a baseline for LLM performance on this task and concludes that future progress is most dependent on developing more efficient methods for automated claim verification.

7.1 Future Work

Building on our findings, several interesting avenues for future research emerge.

- **Scaling search algorithms:** The most critical bottleneck to the systems proposed in this work was the difficulty in scaling up the verification task. Future work should focus on developing optimized and specialized search algorithms. This could involve creating hybrid neuro-symbolic systems where an LLM actively guides a symbolic search process, rather than merely providing a static scoring function to a general-purpose algorithm like DCEM.

- **Refinement of LLM Code Generation:** Our pipeline included a manual debugging portion to prevent runtime errors, and our analysis exposed structural issues with the LLM-generate scoring functions. Refining these processes could greatly improve these systems.

- **Domain expansion:** Our methodologies can be applied to other areas of mathematics such as combinatorics and number theory in order to benchmark LLM performance on a variety of mathematical disciplines.

- **The abilities of open-source models:** Our results demonstrated that open-source models severely lack graph reasoning abilities that many closed source models appear to have. Future work could more closely examine this discrepancy and determine why closed source models are this way and how to improve their counterexample generation abilities.

References

1996. Mathematica. In C. Abbott Paul, editor, *Revival: The Handbook of Software for Engineers and Scientists (1995)*. CRC Press.

Kenneth I. Appel. 1984. The Use of the Computer in the Proof of the Four Color Theorem. *Proceedings of the American Philosophical Society*, 128(1):35–39.

Alan Bundy. 1988. The use of explicit plans to guide inductive proofs. In *9th International Conference on Automated Deduction*, pages 111–120, Berlin, Heidelberg. Springer.

François Charton, Jordan S. Ellenberg, Adam Zsolt Wagner, and Geordie Williamson. 2024. PatternBoost: Constructions in Mathematics with a Little Help from AI. *Preprint*, arXiv:2411.00566.

Jake Chuharski, Elias Rojas Collins, and Mark Meringolo. 2024. Mining Math Conjectures from LLMs: A Pruning Approach. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. *Preprint*, arXiv:2110.14168.

N. G. de Bruijn. 1983. AUTOMATH, a Language for Mathematics. In Jörg H. Siekmann and Graham Wrightson, editors, *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, pages 159–200. Springer, Berlin, Heidelberg.

Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg. Springer-Verlag.

Leonardo De Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob Von Raumer. 2015. The Lean Theorem Prover (System Description). In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25*, volume 9195, pages 378–388. Springer International Publishing, Cham.

Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, Caroline Falkman Olsson, Jean-Stanislas Denain, Anson Ho, Emily de Oliveira Santos, Olli Järviniemi, Matthew Barnett, Robert Sandler, Matej Vrzala, Jaime Sevilla, Qiuyu Ren, Elizabeth Pratt, Lionel Levine, Grant Barkley, and 5 others. 2024. FrontierMath: A Benchmark for Evaluating Advanced Mathematical Reasoning in AI. *Preprint*, arXiv:2411.04872.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, pages 11–15, Pasadena, CA USA.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring Mathematical Problem Solving With the MATH Dataset. In *Thirty-Fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Kotaro Komatsu. 2010. Counter-examples for refinement of conjectures and proofs in primary school mathematics. *The Journal of Mathematical Behavior*, 29(1):1–10.

Yinghui Li, Jiayi Kuang, Haojing Huang, Zhikun Xu, Xinnian Liang, Yi Yu, Wenlian Lu, Yangning Li, Xiaoyu Tan, Chao Qu, Ying Shen, Hai-Tao Zheng, and Philip S. Yu. 2025. One Example Shown, Many Concepts Known! Counterexample-Driven Conceptual Reasoning in Mathematical LLMs. *Preprint*, arXiv:2502.10454.

Pan Lu, Liang Qiu, Wenhao Yu, Sean Welleck, and Kai-Wei Chang. 2023. A Survey of Deep Learning for Mathematical Reasoning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14605–14631, Toronto, Canada. Association for Computational Linguistics.

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. 2023. Self-Refine: Iterative Refinement with Self-Feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594.

Brendan D. McKay and Adolfo Piperno. 2014. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112.

Seyed Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2024. GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models. In *The Thirteenth International Conference on Learning Representations*.

A. Newell and H. Simon. 1956. The logic theory machine–A complex information processing system. *IRE Transactions on Information Theory*, 2(3):61–79.

Stanislas Polu and Ilya Sutskever. 2020. Generative Language Modeling for Automated Theorem Proving. *Preprint*, arXiv:2009.03393.

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. 2025. DeepSeek-Prover-V2: Advancing Formal Mathematical Reasoning via Reinforcement Learning for Subgoal Decomposition. *Preprint*, arXiv:2504.21801.

Alexandre Riazanov and Andrei Voronkov. 2001. Vampire 1.1. In *Automated Reasoning*, pages 376–380, Berlin, Heidelberg. Springer.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2018. Analysing Mathematical Reasoning Abilities of Neural Models. In *International Conference on Learning Representations*.

Shiven Sinha, Shashwat Goel, Ponnurangam Kumaraguru, Jonas Geiping, Matthias Bethge, and Ameya Prabhu. 2025. Can Language Models Falsify? Evaluating Algorithmic Reasoning with Counterexample Creation. *Preprint*, arXiv:2502.19414.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Adam Zsolt Wagner. 2021. Constructions in combinatorics via neural networks. *Preprint*, arXiv:2104.14516.

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep Neural Solver for Math Word Problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 845–854, Copenhagen, Denmark. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

D.H. Wolpert and W.G. Macready. 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. 2024. Qwen2.5-Math Technical Report: Toward Mathematical Expert Model via Self-Improvement. *Preprint*, arXiv:2409.12122.

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J. Prenger, and Animashree Anandkumar. 2023. LeanDojo: Theorem Proving with Retrieval-Augmented Language Models. *Advances in Neural Information Processing Systems*, 36:21573–21612.

A Figures

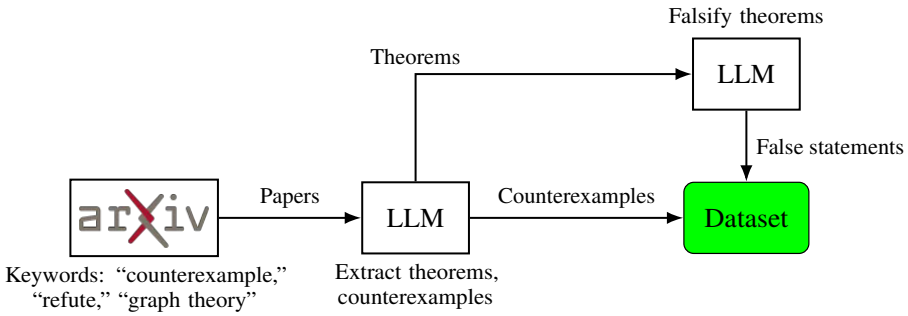

Figure 4: Attempted to scrape counterexamples and theorems from recent papers

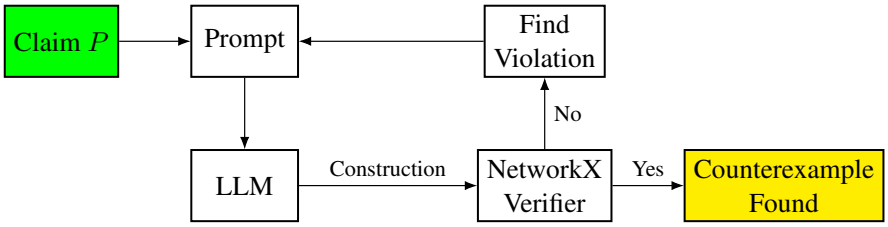

Figure 5: LLM Prompting Pipeline with Refinement Loop

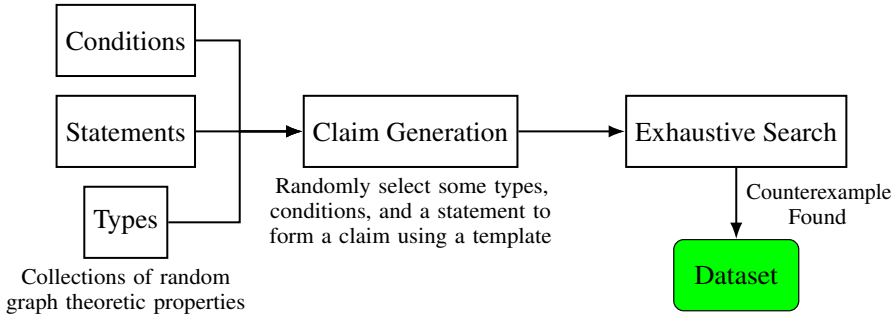

Figure 6: Generating random statements and verifying that they are false with an exhaustive search.

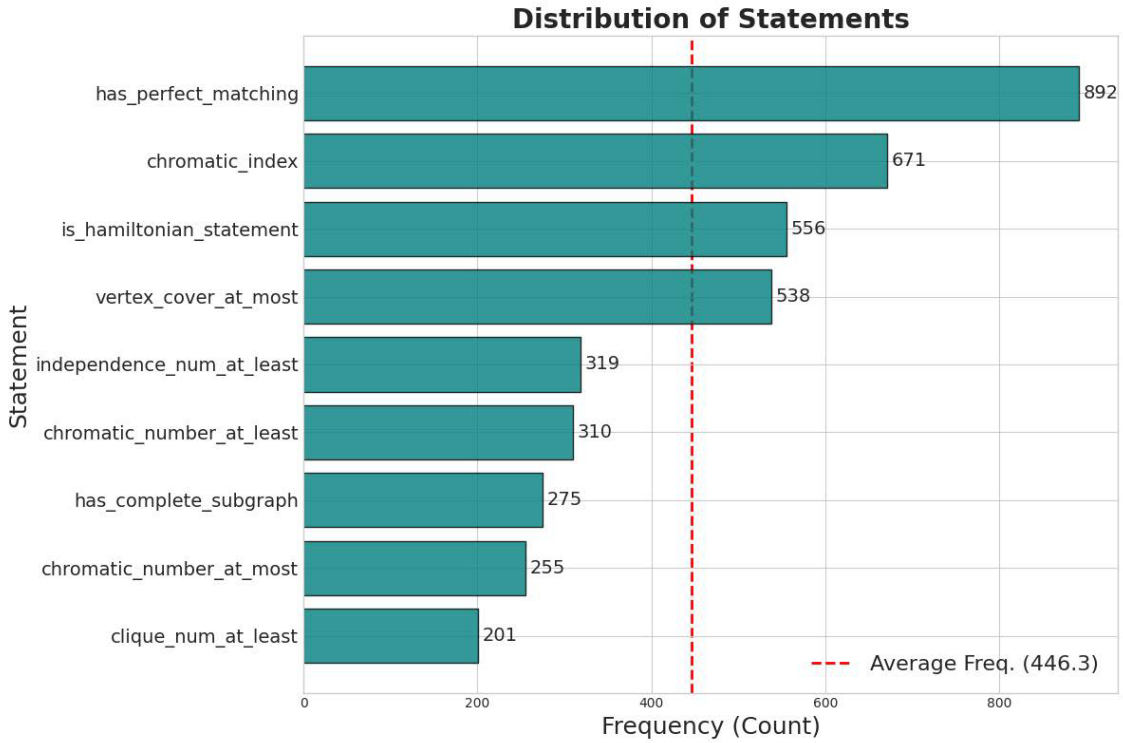

Figure 7: Distribution of Statements in the Current Iteration of the Dataset

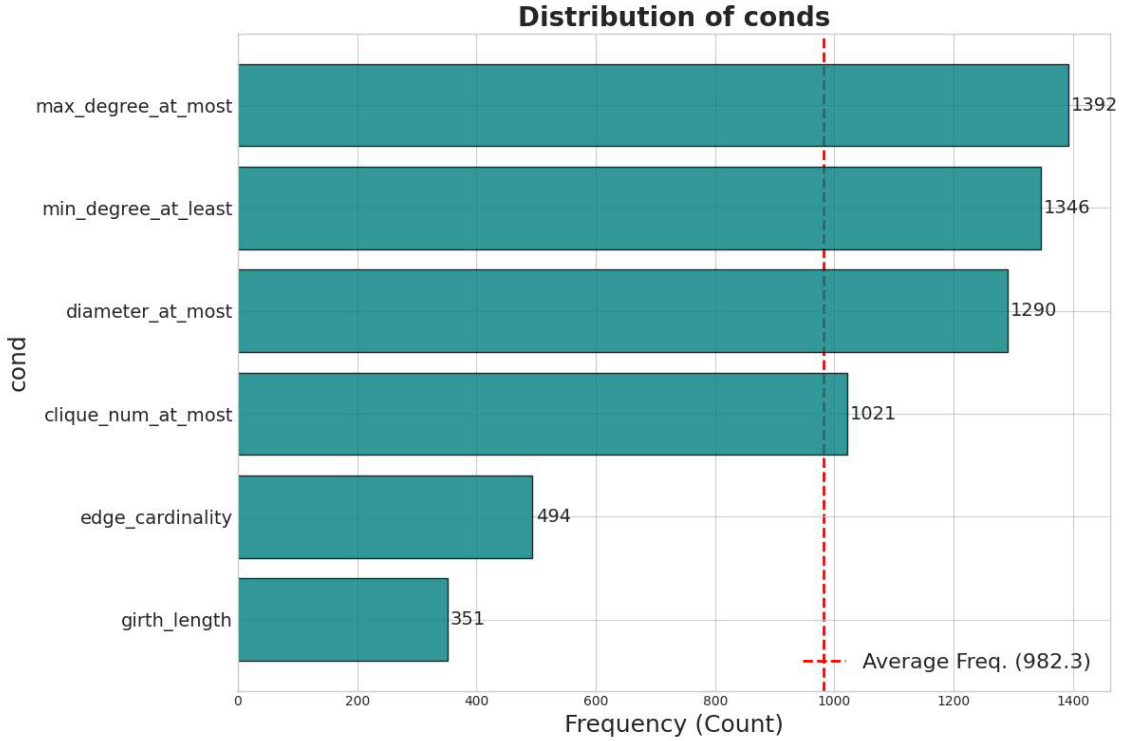

Figure 8: Distribution of Conditions in the Current Iteration of the Dataset

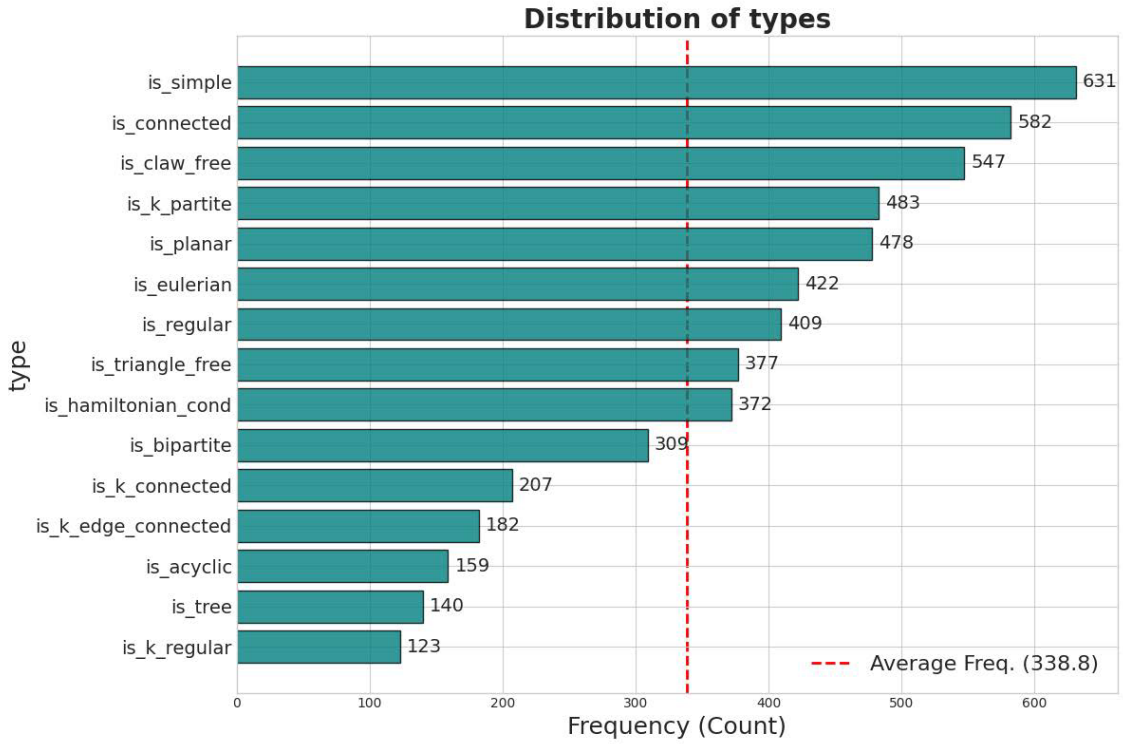

Figure 9: Distribution of Types in the Current Iteration of the Dataset

B LLM Claim Generation Prompt

```
## Role
You are an expert mathematician and creative graph theorist. Your task is to generate a plausible-
    sounding but **false** mathematical claim (a conjecture) in graph theory based on a given theme.
     The claim should be specific, non-trivial, and sound like a real mathematical conjecture.
    Claims should have a relatively large minimum counterexample if possible (10-15 vertices). Avoid
     claims with very small counterexamples (<6-7 vertices). Additionally, each part of the claim (
    each condition and the statement) should be relatively easily verified so an automated
    counterexample search can take place. Ensure all mathematical notation uses LaTeX commands, not
    unicode.

## Framework:
Your response must be structured according to the following formal framework:

A claim, denoted as P, is a pair (s, C).

C is a set of conditions. A condition is a property that a graph x must satisfy.

s is a statement. The statement is a property that is asserted to be true for any graph x that meets
    all the conditions in C.

### Example:
If the input theme is "Hamiltonian Paths," a valid claim is "All connected graphs contain a Hamilton
    cycle." Here is how you would break it down:

Claim: All connected graphs contain a Hamilton cycle.

Conditions (C):

The graph is connected.

Statement (s):

The graph contains a Hamilton cycle.
```

```
## Your Task:
Generate a false, non-trivial graph-theoretic claim on undirected graphs related to the user provided
     theme. Reason through the claim generation process step by step in order to generate a
    realistic claim. Provide the full claim in natural language, and then explicitly list the
    conditions (C) and the statement (s).

Additionally, there are two tasks you must complete once you have created your claim.
1. You must judge the difficulty of finding a counterexample to the claim on a scale of 1 (very easy)
     to 5 (very difficult). Once again, reason through the grading process in order to generate an
    accurate difficulty score.
2. You must determine a reasonable range for the number of vertices on which a counterexample may
    exist. Use your knowledge of graph theory to reason through possible sizes and output 5 possible
     values.

## Output:
Your final response should be in the following JSON schema.

‘‘‘json
{
   "claim": str,
   "conditions": list[str],
   "statement": str,
   "score": int,
   "counterexample_vertices": list[int]
}
‘‘‘
```

C LLM Predicate Generation Prompt

```
## Role
You are an expert in graph theory and a skilled Python programmer with deep knowledge of the ‘
    networkx‘ library. Your task is to translate a given graph-theoretic property into a Python
    predicate function that evaluates that property on a given graph.

## Framework
You will be provided with a single property of a graph, potentially containing LaTeX notation. Your
    goal is to create a Python function that takes a ‘networkx‘ graph object, ‘G‘, as input and
    returns ‘True‘ **only** if ‘G‘ satisfies the property, and ‘False‘ otherwise. There must be no
    false positives or false negatives.

### Instructions:
1.  **Function Definition:** Give the function a descriptive name. The name of all helper functions
    must be ‘helper_...‘. It must accept **only** a single argument, ‘G‘, which will be a ‘networkx‘
     graph object.
2. **Planning:** Before writing any Python code, use your knowledge of graph theory to plan out the
    structure of the function.
3.  **Implementation:** The function’s logic must accurately check for the given graph property. You
    should prioritize using the ‘networkx‘ library’s built-in functionalities for efficiency and
    correctness (Note: ‘networkx‘ does **not** contain a ‘chromatic_number‘, ‘is_k_connected‘ or ‘
    is_hamiltonian‘ function, but does contain helpful functions such as ‘girth‘, ‘is_regular‘, ‘
    adjacency_spectrum‘, ‘node_connectivity‘ etc.). You also have access to the ‘numpy‘ and ‘scipy‘
    library. The code should be self-contained and ready to execute, including any necessary imports
     from ‘networkx‘, ‘numpy‘ or ‘scipy‘. Ensure that all functions needed are imported and used
    correctly (Note: ‘networkx‘ has functions which are only accessible from submodules such as ‘nx.
    approximation.max_clique‘).

## Input Example:
You will receive the property as a single natural language string. Ex.

‘The graph is connected.‘

## Output Structure:
Your final output must adhere strictly to the following structure. Include very few comments other
    than a docstring.

**Python Predicate Function:**
‘‘‘python
[Your complete Python predicate function here.]
‘‘‘
```

D LLM Scoring Function Generation Prompt

```
## Role
You are an expert in graph theory and a skilled Python programmer with deep knowledge of the ‘
    networkx‘ library. Your task is to translate a given graph-theoretic claim into a **Python
    scoring function** for the purpose of *Counterexample Search*. This function will evaluate how
    close a given graph is to being a counterexample to the claim.

## Framework
You will be provided with a single natural language graph-theoretic claim, potentially containing
    LaTeX notation. Your goal is to create a Python function that takes a ‘networkx‘ graph object, ‘
    G‘, as input and returns a **fine-grained score as a float** based on closeness to being a
    counterexample. A graph is a counterexample of a claim if it satisfies all conditions, but
    violates the statement. The score should be in the interval [0, infinity).

A score **>= 1.0** must mean the graph is very likely a true counterexample to the claim. A score
    less than 1.0 should quantify how "close" the graph is to being a counterexample, with values
    close to and above 1.0 should indicate greater closeness, and scores closer to 0 should indicate
     that the graph is very far from being a counterexample to the claim.

### Instructions:
1.  **Function Definition:** You must write a single function, helper functions are allowed. Give the
     functions descriptive names. The names of all helper functions must start with ‘helper_‘. The
    main scoring function must accept **only** a single argument, ‘G‘, which will be a ‘networkx‘
    graph object, and it must return a ‘float‘.
2.  **Devise a Metric:** Before writing any Python code, use your knowledge of graph theory to plan
    the scoring logic. Think critically about how to measure "closeness" for the given claim. For
    example, if the claim states the graph is connected, closeness could be inversely related to the
     number of connected components. During this step, also consider efficiency. This function will
    be called many times, so avoid exponential time algorithms.
3.  **Implementation:** The function’s logic must accurately implement the scoring metric you devised.
     You should prioritize using the ‘networkx‘ library’s built-in functionalities for efficiency
    and correctness (Note: ‘networkx‘ does **not** contain a ‘chromatic_number‘, ‘is_k_connected‘ or
     ‘is_hamiltonian‘ function, but does contain helpful functions such as ‘girth‘, ‘is_regular‘, ‘
    adjacency_spectrum‘, ‘node_connectivity‘, etc.). The code should be self-contained and ready to
    execute, including any necessary imports from ‘networkx‘, ‘scipy‘ or ‘numpy‘. Ensure that all
    functions needed are imported and used correctly (Note: ‘networkx‘ has functions which are only
    accessible from submodules such as ‘approximation.max_clique‘).

## Input Example:
You will receive the claim as a single natural language string. Ex.

‘All connected graphs contain a Hamilton cycle.‘

## Output Structure:
Your final output must adhere strictly to the following structure. Include very few comments other
    than a docstring.

**Python Scoring Function:**
‘‘‘python
[Your complete Python scoring function here.]
‘‘‘
```

E LLM Counterexample Generation Prompt

```
You are an LLM assistant which specializes in graph theory.
Your task is to construct **explicit counterexamples** to user-supplied claims about finite
    undirected graphs and to return each counterexample solely as an adjacency-matrix representation.


---

1. **Input you will receive**

   * Input will be given in the following format:
     * "Let $G$ be a [CONDITIONS] graph on $x\\leq n \\leq X$ vertices such that [CONDITIONS]. Then
         [STATEMENT]."
   * A claim is a set of conditions and a statement. A counterexample to a claim satisfies all
       conditions but violates the statement.
```

```
2. **Parse the claim**

   * Identify all conditions asserted (order, connectivity, planarity, Hamiltonicity, chromatic
     number, etc.).
   * Identify the meaning of all notation used.
   * Identify the statement and negate it to determine what a counterexample looks like.

3. **Search for a Counterexample**

   * **Leverage Known Graphs:** Consider well-known graphs that are famous for being counterexamples
     or having specific properties.
     * **Complete Graphs ($K_n$):** Useful for claims involving cliques, colorings, and edge density.

     * **Complete Bipartite Graphs ($K_{m,n}$):** Especially the "star graphs" ($K_{1,n}$) and the
       utility graph ($K_{3,3}$). These are critical for testing claims about bipartiteness,
       planarity, and cycles.
     * **Cycle Graphs ($C_n$):** Testbeds for claims about connectivity, Hamiltonicity, and
       chromatic number. Odd cycles are not bipartite.
     * **Path Graphs ($P_n$) and Wheel Graphs ($W_n$):** Simple structures for testing basic
       properties.
     * **Platonic Solid Graphs:** The graphs of the tetrahedron, cube, octahedron, dodecahedron, and
       icosahedron are highly symmetric and have distinct properties related to regularity and
       connectivity.

   * **Constructive Approach:** If the claim involves multiple conditions (e.g., "bipartite and 3-
     regular"), try to build a graph that meets them. Start with a graph that satisfies the most
     restrictive condition and incrementally modify it to satisfy the others. During this process,
     constantly check if the statement is being violated.

   * **Heuristic Checks:**
     * **Connectivity:** Does removing a vertex or edge disconnect the graph? Check for cut vertices
       and bridges.
     * **Degree Sequence:** What do the degrees of the vertices tell you? For instance, Dirac's
       theorem gives a sufficient condition for a Hamiltonian cycle based on minimum degree. If
       the claim is about Hamiltonicity, look for a graph that *just* fails to meet this condition.

     * **Girth:** What is the length of the shortest cycle? This is crucial for claims about
       bipartiteness (a graph is bipartite if and only if it has no odd cycles).
     * **Planarity:** Can the graph be drawn without edge crossings? Test for minors of $K_5$ and
       $K_{3,3}$ (Kuratowski's theorem).

   * By systematically applying these search strategies, you will increase the likelihood of
     efficiently identifying a valid counterexample. Always prioritize the smallest valid
     counterexample you can find.

4. **Verify correctness**

   * Internally prove to yourself (e.g., by reasoning or quick algorithmic checks) that
     * All stated conditions are met, and
     * The statement fails for this graph.
   * If every finite graph satisfying the constraints fulfills the claim, output "No counterexample
     exists."

5. **Prepare output**

   * Convert the counterexample graph found to an adjacency matrix.
   * An adjacency matrix is a matrix with dimensions $n \times n$ where $n$ is the number of vertices
     in the graph, and the $i,j$-th entry in the matrix is a 1 if there is an edge between vertex
     $i$ and vertex $j$ and is 0 otherwise.

---

Use these rules exactly to generate adjacency-matrix counterexamples for any graph-theory claim the
    user supplies. Please reason step by step, and put your final answer within \\boxed{}.
```