# COE 379L Project 4 Report: LLMs for College Level Math

Tanya Jagan

## INTRODUCTION

Large Language Models (LLMs) such as Chat GPT, Claude, and Gemini are becoming increasingly used for question answering, with one third of college students reporting that they consult LLMs for their coursework. While LLMs have demonstrated fairly acceptable accuracy with simple questions, the use of LLMs for answering questions in higher academic and research settings can be risky. LLMs are known to sometimes produce factually incorrect or misleading information with high confidence, raising significant concerns when used as educational or scholarly tools.

This project is inspired by the TruthfulQA benchmarking project and aims to assess LLMs' ability to answer higher level math questions. The models will be probed without providing explicit examples. Trends in model errors and limitations will be analyzed.

## DATA SOURCES

The following section describes the Omni-MATH dataset and the subset of the dataset that was used for this project.

### *Omni-MATH Dataset*

The Omni-MATH dataset consists of 4428 competition-level math problems. This dataset was selected over other math QA datasets due to its range of questions in terms of difficulty and topic as well as its detailed annotations. For each problem, there is an associated category, difficulty level (1-10), solution, answer, and source. Within the dataset, there are subdomains such as Algebra, Discrete Math, Geometry, and Calculus. Within each of these subdomains exist subsubdomains such as Prealgebra, Integral Calculus, Plane Geometry, and Graph Theory. In total, there are nine subdomains and 33 subsubdomains. Difficulty level was determined according to the Art of Problem Solving's website's difficulty levels for problems listed there. Problem statements and detailed solutions were collected from numerous international competitions and converted into LaTeX format. These competitions vary in difficulty, including introductory level competitions (e.g. CEMC: Pascal) and internationally recognized difficult competitions (e.g. Putnam). Table 1 below shows an example data point from Omni-MATH.

Table 1. Omni-MATH Example Data Point

| | |
|---|---|
| Domain | Mathematics -> Discrete Mathematics -> Graph Theory |
| Difficulty | 6.5 |
| Problem Source | china_team_selection_test |
| Question | Fix positive integers k,n. A candy vending machine has many different colours of candy, where there are 2n candies of each |

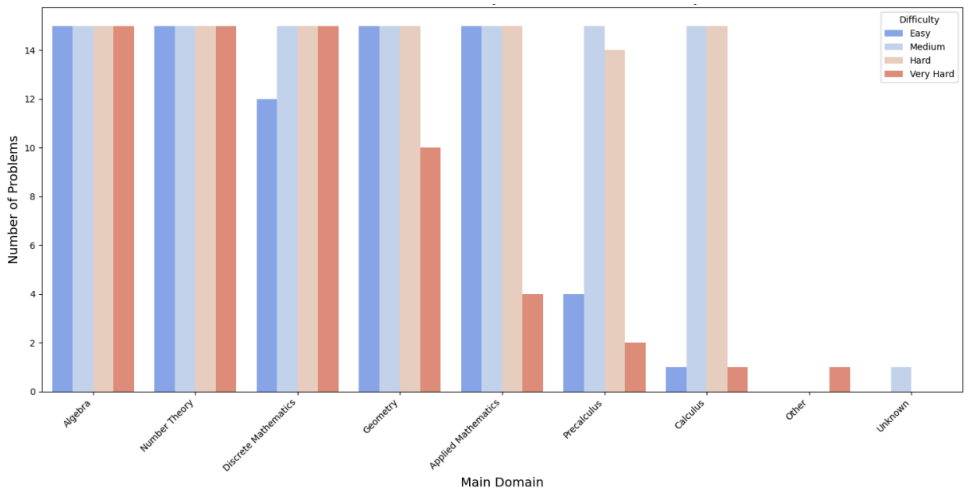
| | |
|---|---|
| | colour. A couple of kids each buys from the vending machine 2 candies of different colours. Given that for any k+1 kids there are two kids who have at least one colour of candy in common, find the maximum number of kids. |
| Solution | Fix positive integers k and n. Consider a candy vending machine that has many different colors of candy, with 2n candies of each color. A couple of kids each buys from the vending machine 2 candies of different colors. We are to find the maximum number of kids such that for any k+1 kids, there are two kids who have at least one color of candy in common. We can model this problem using graph theory. Let each kid be represented by a vertex, and draw an edge between two vertices if the corresponding kids share a color of candy. The problem then reduces to finding the maximum number of vertices in a graph such that any k+1 vertices have at least one edge connecting them. This is equivalent to finding the maximum number of vertices in a graph where the largest independent set has size k. By Turán's theorem, the maximum number of vertices in such a graph is given by the Turán number T(n,k+1), which is the maximum number of edges in a k-partite graph with n vertices in each part. The answer is n(3k). Thus, the maximum number of kids is: n(3k). |
| Answer | n(3k) |

### *Subset Construction and Preprocessing*

Due to computing limitations, a subset of 350 questions was constructed from the Omni-MATH dataset and additional fields were derived to facilitate stratified sampling. The *domain* annotation was parsed to extract *main_domain* and *subsub_domain* for each problem and problems were categorized into discrete difficulty bins: Easy (1-3), Medium (4-6), Hard (7-8), and Very Hard (9-10).

A stratified sampling procedure was implemented to ensure balanced coverage across domain and difficulty bin. For each (*main_domain*, *difficulty_bin*) pair, up to 15 problems were randomly selected. Distribution of the subset across domains and difficulty bins is pictured in Figure 1 below. This subset was used for the project.

Figure 1. Distribution of Problems by Main Domain and Difficulty



## METHODOLOGY

The following section describes how the LLMs were prompted and various evaluation strategies.

### *Prompting the LLM With Questions*

#### *1.1 Preprocessing For Formatting*

To reduce format-driven variability during inference, LaTeX math formatting was stripped and replaced with plain mathematical text. However, due to the diversity of QA text within the dataset as well as LLM responses, the conversion method often did not catch every instance. However, when LaTeX formatted questions versus plain text questions were fed into the LLMs, answers did not vary significantly; as such, LaTeX formatting was kept.

#### *1.2 Model Inference*

Problems were posed to two models: OpenAI GPT-4o-mini and Gemini. Both models were queried in zero-shot settings using prompt templates designed to simulate a student's question and to encourage reasoning and a clear final answer. For each batch of two problems, the prompt was as follows: *"I have a few math questions. For each one, please give a detailed explanation first, and then clearly write 'Final Answer: <your answer>' on its own line."* A small batch size was used due to token limitations, and a high *max_tokens* of 3000 was set to ensure multi-step solutions for both questions. *Temperature* was set to 0.0 to minimize stochasticity.

#### *1.3 API Automation*

ThreadPoolExecutor was used to run parallel batches, which enabled faster throughput while staying within the rate limits. Each output was then parsed to get the solution until the model output the string *"Final Answer:"*, associated each answer with its preceding explanation, and then stored the data to a CSV.

#### *1.4 Response Formatting Challenges*

Despite addressing formatting in prompting, response formatting varied slightly across batches. In some responses, the final answer was omitted or formatted inconsistently, long explanations sometimes caused truncation, and some answers were embedded mid-paragraph instead of on a separate line. To mitigate this, the answer results were manually reviewed and corrected.

### *Evaluation Procedure*

Given that there were a variety of response types – qualitative and quantitative solutions and answers – multiple strategies were employed to evaluate correctness.

#### *2.0 Final Answer Matching*

A simple string comparison was used to determine if the model's final answer matched the reference answer. This approach worked well when there was a single numerical answer; however, if the answer contained variations in formatting or had a qualitative nature, correct answers were often marked incorrect. As such, final answer matching was insufficient.

#### *2.1 Embedding Similarity via MathBERT*

Semantic alignment between model and reference solutions was explored using the tbs17/MathBERT model, which is trained on math text and supports contextual embeddings. The solution responses were encoded using the CLS token representation model, and cosine similarity was computed between embeddings. The standard similarity threshold of 0.70 was used as an initial cutoff for comparisons. While this method occasionally showed different but valid solutions that string matching would miss, several limitations were found in this method as well.

MathBERT embeddings often reflected language co-occurrence patterns and statistical associations rather than deeper semantic and logical structure. Incorrect solutions that still used the same keywords, variables, or expressions (which were sometimes simply drawn from the problem) as the reference solutions scored similarly to correct solutions. Moreover, even in these cases the cosine similarity was often well below the threshold, at around <0.50. Also, short solutions that are still correct tended to have less contextual content, which could reduce their embedding richness, thus leading to lower similarity scores than longer, flawed solutions.

Table 2.0 Cosine Similarity for Correctness vs Incorrectness

| | Model Is Correct | Model is Incorrect |
|---|---|---|
| Reference Answer | (1998/1997)^2 | y = sqrt(2) |
| Model Answer | (1998/1997)^2 | y = 1 |
| Cosine Similarity For Solutions | 0.4649 | 0.4710 |

### *2.2 Sentiment Analysis*

An exploratory attempt was made to score solutions based on sentiment and confidence; however, models often replied to questions with high confidence regardless of question difficulty and response accuracy. As such, sentiment analysis was not further applied.

### *2.3 LLM As Judge*

Due to limitations with previous NLP methods, the LLM-as-judge method was adopted. Unlike the previous models, GPT 4o is better suited for evaluating reasoning and accounting for slight variations in responses. The prompt was as follows: “I am a math teacher. Given a math problem, a student’s solution, and the reference solution, determine if the student’s final solution is correct.” The LLM was able to manage formatting differences such as sqrt(144) vs 12 and LaTeX formatting. The LLM’s judgment is still based on NLP methods rather than mathematical proofs, but serves as the best automated correctness evaluator compared to human judgment.

## FINDINGS

This section outlines observed performance trends across models, problem categories, and difficulty levels, using correctness from GPT 4o.

***Accuracy***

Both models had low accuracy on the dataset, with GPT-4o-mini slightly outperforming Gemini, especially in structured procedural domains such as algebra and precalculus. The accuracy scores reflect the difficulty and formality of the dataset—many problems required multi-step reasoning, abstract generalization, or symbolic manipulation, which LLMs are not fully optimized for.

| Model | Accuracy (%) |
|---|---|
| GPT-4o-mini | **38.0** |
| Gemini | **34.0** |

As expected, accuracy dropped as problem difficulty increased, consistent with prior work in LLM math evaluation. Easier problems were often solved correctly with basic formula recall or simple computation, while higher difficulty problems required multi-variable reasoning, proof steps, or abstract logic.

| Difficulty Bin | GPT-4o-mini (%) | Gemini (%) |
|---|---|---|
| Easy (1–3) | 72.1% | 66.8% |
| Medium (4–6) | 38.3% | 35.2% |
| Hard (7–8) | 19.7% | 17.6% |
| Very Hard (9–10) | 8.2% | 6.5% |

At lower difficulty levels, both models frequently produced correct answers. These problems typically involved direct applications of known formulas (e.g., Pythagorean Theorem, basic probability) and required few inferential steps. As difficulty increased, accuracy declined sharply. Hard and Very Hard problems often required symbolic transformations, several variables, abstract reasoning, and other advanced concepts.

| Domain | GPT-4o-mini (%) | Gemini (%) |
|---|---|---|
| Algebra | 47.2 | 42.7 |
| Precalculus | 43.4 | 40.4 |
| Geometry | 36.2 | 33.8 |
| Number Theory | 34.2 | 30.7 |
| Discrete Math | 24.3 | 19.4 |
| Applied Math | 35.6 | 31.2 |

Algebra and precalculus exhibited the highest performance. These domains follow standard solution templates, and rely on arithmetic manipulation and formula recall—strengths of large-scale language models.

In contrast, performance in discrete mathematics was significantly lower. Problems in this category typically require combinatorial reasoning, casework, counting arguments, or graph-theoretic logic—areas where language models tend to hallucinate intermediate steps or apply probabilistic heuristics.

Number theory and geometry presented challenges due to their symbolic density and reliance on abstract reasoning (e.g., modular arithmetic, angle chasing, geometric constructions). While GPT-4o-mini handled numeric examples more consistently, it struggled to generalize symbolic reasoning patterns.

### *Common Error Patterns*

Manual review of errors revealed common reasons for failure.

1. Pattern Memorization: Both models often attempted to recall familiar solution patterns, even when the problem required a novel approach. For example, applying the quadratic formula to non-quadratic expressions, or using standard triangle properties in non-Euclidean contexts.
2. Symbol Confusion: Variables and operators from the problem prompt were sometimes carried into the solution when they should not have been, suggesting copying rather than comprehension when attempting a solution.
3. Generalization: In problems requiring induction or bounding arguments, models sometimes generalized incorrectly, asserting unproven patterns or making invalid assumptions.
4. Inconsistent Units: LLMs sometimes manipulated numbers correctly but ignored or mismatched units in intermediate or final steps.
5. Early Truncation: In longer derivations, particularly for multi-case problems, the models occasionally truncated responses or hallucinated conclusions inconsistent with prior logic.

## CONCLUSION

This project affirms that while GPT-4o-mini and Gemini are capable of solving some college-level math problems, they lack the consistency and precision required for advanced academic applications. GPT-4o-mini performed better than Gemini, especially in algebraic and medium-difficulty problems, but still struggled with symbolic abstraction and logical reasoning in harder problems.

These findings align with the broader observation that current LLMs are unreliable for formal mathematical reasoning without some sort of external verification. Further advancements in multi-step reasoning and training on higher level math are likely needed to improve performance on tasks of this nature.

## REFERENCES


1. Dataset: https://omni-math.github.io/
2. TruthfulQA: https://github.com/sylinrl/TruthfulQA
3. NLP Resources: https://coe379l-sp25.readthedocs.io/en/latest/unit04/overview.html