Solving Mathematical Problems using Large Language Models: A Survey

Feijuan He[1,4], Han Lai[2†], Jiaqi Liu[3], Bo Wang[1], Haoran Chen[1], Haohan Liu[1], Chenxi Zhang[1]

[1]Xi'an Jiaotong University City College, Shaanxi 710018, China

[2]School of Computer Science and Technology, Xi'an Jiaotong University, Shaanxi 710049, China

[3]Dongfang Hospital, Beijing University of Chinese Medicine, Beijing 100078, China

[4]Engineering research center of IoT intelligent sensing interactive platform, universities of Shaanxi province, Shaanxi 710018, China

Keywords: Large language models; Mathematical problems; Fine-tuning; Prompt engineering; Symbolic solvers; Evaluators; Validators; Survey; Knowledge graph

Citation: He F.J., Lai H., Liu J.Q., et al.: Solving Mathematical Problems using Large Language Models: a Survey. Data Intelligence, Vol. 7, Art. No.: 2025r44, pp. 1-40, 2025. DOI: https://doi.org/10.3724/2096-7004.di.2025.0064

ABSTRACT

Large Language Models (LLMs) exhibit impressive performance across various Natural Language Processing (NLP) tasks due to their robust contextual understanding, content generation and few/zero-shot learning abilities. However, LLMs still show significant limitations while handling with mathematical problems that require complex reasoning skills and interpretable solving processes. Consequently, a series of research efforts have been made in solving mathematical problems using Large Language Models (SMP-LLM). This survey provides a comprehensive review of such endeavors. First, we introduce a two-layer classification system for SMP-LLM: at the primary layer, we categorize extant researches into four classes of solution methods, including: fine-tuning, prompt engineering, collaboration with symbolic solvers, and collaboration with evaluators/validators. At the second layer, we classify mathematical problems into four categories: math word problem, geometry problem, theorem proving, and combinatorial optimization problem. This classification system finds the correlation between solution methods and the categories of mathematical problems. Second, we analyzed typical research works under of each method, and summarized their strengths and weaknesses. Third, we elucidate current mainstream datasets for solving mathematical problems and analyzed how these datasets promote SMP-LLM research from different perspectives. Finally, summarize the challenges that SMP-LLM are facing and highlighted four research directions: geometric analysis, comprehension, and generation of mathematical expressions, indirect reasoning and benchmarks for evaluating mathematical ability. We hope that this survey can provide useful references for researchers interested in SMP-LLM.

† Corresponding author: Han Lai (E-mail: hanlai@stu.xjtu.edu.cn; ORCID:0009-0000-3310-8075).

1. INTRODUCTION

Large Language Models (LLMs) refer to deep learning models trained on massive text datasets using self-supervised learning, with parameter sizes typically reaching billions or higher. LLMs possess strong capabilities in contextual understanding, content generation, and few/zero-shot learning, they can adapt to various natural language understanding and generation tasks with minimal or no fine-tuning. They have achieved state-of-the-art performance in tasks such as machine translation, text summarization, sentiment analysis, and even reached human-level performance. However, LLMs usually employ autoregressive methods for training and next-token prediction methods for content generation, which leads to weaker performance in tasks requiring numerical reasoning, symbolic reasoning, and logical reasoning. At the same time, increasing the number of parameters of the model can not significantly improve its reasoning performance [1] and may arise inverse scaling phenomena in tasks such as modus tollens inference [2].

Utilizing LLMs directly during mathematical problem solving is challenging because various types of reasoning tasks are involved. Therefore, the ability to solve mathematical problems is an important indicator in evaluating intelligent systems. A series of evaluations on LLMs have also confirmed this conclusion [1, 3-9].

For this purpose, a series of researches have been made on mathematical problem solving with open-source or closed-source LLMs. These research efforts target various types of mathematical problems, including math word problem, geometry problem, theorem proving and combinatorial optimization problem. They employed diverse technical approaches such as fine-tuning, prompt engineering, collaboration with symbolic solvers and collaboration with evaluators/validators to enhance LLMs' ablility on solving complex mathematical problems. This survey will provide a review over such endeavors. Currently, we have only find one relevant survey [10] which conducts a review of researches on mathematical language models from two perspectives: mathematical problem types and solution methods. The former mainly encompasses arithmetic operations and mathematical reasoning, while the latter is divided into two major categories: based on pre-trained language models and based on LLMs. While this survey provides valuable reference and guide for researchers engaged in solving mathematical problems with LLMs, it still has two shortcomings. First, its classification system is relatively ambiguous. In a good classification system, each category should be *mutually exclusive, unambiguous* [11]. However, in the classification system of [10], the boundary between pre-trained language models and LLMs is blurred, and the boundary between math word problems and mathematical question-answering problems is also unclear. Thus, it fails to demonstrate the essential characteristics of different types of research work effectively. Second, it fails to reflect that different types of mathematical problems have varying levels of difficulty, they rely on different mathematical knowledge, and require different solution strategies. Therefore, there is strong relevance between mathematical problem types and solution methods, informing readers about this correlation is helpful in deepening their understanding of this field. However [10], introduces relevant research work from the perspectives of mathematical problem types and solution methods independently, which fails in demonstrate their correlation clearly.

Unlike the previous survey, our survey further focuses on the research efforts in solving mathematical problems using LLM, SMP-LLM, and proposes future research directions. The main contributions are reflected in four aspects:

(1) We propose a two-layer classification system for SMP-LLM. At the first layer, existing research is classified into four categories based on the approach to problem-solving, including fine-tuning, prompt engineering, collaboration with symbolic solvers, and collaboration with evaluators/validators. At the second layer, mathematical problems addressed by existing researches are classified into four categories, including math word problem, geometric problem, theorem proving, and combinatorial optimization problem. We present the classification system in a grid format, demonstrating the relationship between solution methods and types of mathematical problems.

(2) We analyze typical research works for each category and summarize the advantages and disadvantages of each type of work based on the proposed classification system.

(3) We exposit current mainstream mathematical problem-solving datasets, and analyze how these datasets promote research in SMP-LLM from various perspectives.

(4) We summarize the challenges SMP-LLM is facing and identify four future research directions: geometric analysis, mathematical expression comprehension and generation, indirect reasoning and the benchmark for mathematical proficiency.

The organization of the remaining sections of the survey is as follows: Section 2 introduces the classification system of SMP-LLM. Sections 3 to 6 respectively review typical research works in the four categories of fine-tuning, prompt engineering, collaboration with symbolic solvers, and collaboration with evaluators/validators. Section 7 analyzes datasets related to mathematical problem solving. Section 8 summarizes the challenges faced and future research directions. Finally, Section 9 presents the conclusions.

2. CLASSIFICATION SYSTEM

2.1 Solution Methods

From the perspective of solution methods, existing researches on SMP-LLM mainly adopts two strategies: one is to enhance or activate the mathematical reasoning ability of LLMs themselves, and the other is to enable LLMs to collaborate with other modules while solving mathematical problems.

The former includes two methods: fine-tuning and prompt engineering. The fine-tuning methods usually adjust small numbers of paramters in pre-trained language models using instruction-formatted instances in supervised learning ways to adapt them to specific tasks. Instruction-formatted instances typically consist of task descriptions (referred to as instructions), input-output pairs, and a small number of optional demonstrations [12]. Prompt engineering usually use well-designed prompts to assist LLMs to better adapt to their tasks, further comprising two subclasses: (1) In-context learning (ICL), which adds

natural language-formatted examples in prefix form for queries [13] to enable LLMs to solve specific types of mathematical problems through analogical reasoning. (2) Chain-of-Thought (CoT) instructs the model to generate a series of interrelated intermediate reasoning steps [13]. CoT significantly improves the multi-step reasoning ability of LLMs, which is a common requirement for solving complex mathematical problems. Unlike fine-tuning methods, prompt engineering will not modify the parameters of LLMs.

The latter also includes two methods: collaboration with symbolic solvers and collaboration with evaluators/validators. The methods of collaborating with symbolic solvers synthesize LLMs' advantages in contextual understanding and symbolic systems' advantages in composability, interpretability and complex reasoning. In the methods of collaboration with evaluators/validators, LLMs usually serve as the generator, executor or enhancer of the solution. In which, evaluators are usually used to assess the correctness of solutions generated by the LLM and select the optimal solution based on the evaluation results, while validators are mainly used to provide feedback to the LLM to help improve its solutions. They enable the LLMs to improve the solution iteratively.

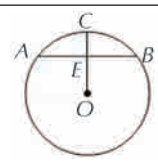
2.2 Mathematical Problem Types

From the perspective of mathematical problem types, the existing researches on SMP-LLMs mainly focuses on four types of mathematical problems:

First, Math Word Problems, which require solving specific real-life problems using mathematical methods. Solving these problems usually involves understanding the problem scenario, extracting key information, establishing mathematical models, and conducting multi-step reasoning to ultimately provide interpretable solutions. An example of math word problems is shown as example 1.

> **Example 1.** If Sally has 5 pencils and her friend gives her 3 more pencils, how many pencils does Sally have in total?

Second, Geometry Problems, which involve mathematical problems related to geometric shapes, spatial relationships and properties. Solving these types of problems requires accurately identifying geometric figures, understanding their spatial relationships, applying various geometric theorems, adding necessary auxiliary lines, and performing logical reasoning and numerical calculations. An example of geometry problems is shown as example 2.

> **Example 2.** As shown in the figure, in ⊙O, AB is the chord, OC ⊥ AB, if the radius of OO is 5 and CE=2 , then the length of AB is ().
> A. 2 B. 4 C. 6 D. 8
>
> 
>

Third, Theorem Proving, which requires finding a direct or indirect method to prove mathematical theorems. Solving these types of problems involves executing logical reasoning and symbolic computation

based on pre-constructed theorem library, and effectively controlling the reasoning path and search space to find the optimal proof path. An example of theorem proving is shown as example 3.

> **Example 3.** Prove the theorem that there are infinitely many prime numbers.

Forth, Combinatorial Optimization Problems, which requires finding the optimal combination within a set of constraints to maximize or minimize a certain objective function. Combinatorial optimization problems face the challenge of combinatorial explosion. On one hand, efficient search algorithms are needed to find the optimal solution in a vast search space. On the other hand, need to maintain diversity in the search to avoid falling into a local optima. An example of combinatorial optimization problems is shown as example 4.

> **Example 4.** Online Bin Packing Problem: [14]
> **Input**: Bins with limited capacity and arriving items with varying sizes.
> **Constraints**: (1) Each item must be assigned to a bin upon arrival. (2) Once an item is assigned to a bin, it cannot be moved to a different bin. (3) The total size of items in a bin cannot exceed the bin's capacity.
> **Objective**: Minimize the number of bins used to pack all items.

Solving the above four types of problems requires various abilities shown in Table 1. Compared to ✓, ✓✓ indicates a higher demand for that ability. But because LLMs possess only language understanding abilities and relatively weaker logical reasoning and numerical computation abilities, researchers need to make more efforts to help LLMs gain the abilities mentioned above.

Table 1. Different Abilities Required by Different Types of Mathematical Problems.

Capacity	Math Word Problem	Geometry Problem	Theorem Proving	Combinatorial Optimization
Text Understanding	✓	✓	✓	✓
Diagram Parsing		✓		
Logical Reasoning	✓	✓✓	✓✓	✓
Numerical Calculation	✓	✓		✓
Symbolic Manipulation		✓	✓✓	✓✓
Spatial Search		✓✓	✓✓	✓✓

2.3 A Two-layer Classification System of SMP-LLM

Combining the above analysis, we propose a two-layer classification system of *Solution Method-Problem Type*, as shown in Figure 1, which demonstrates the correlation between solution methods and mathematical problem types of SMP-LLM. Fine-tuning is suitable only for simple math word problems and geometry problems, as it is limited by the quality of training data and the capability of LLM itself. Though prompt engineering can guide LLMs to generate expected outputs through methods such as Chain-of-Thought, it is still limited by LLM's reasoning abilities and cannot solve mathematical problems involving complex reasoning, so it is mainly used in solving math word problems. Compared to the first two methods, combining LLMs with external modules such as symbolic solvers, evaluators or validators, can better develop the potential of LLMs and enhance their abilities to solve more complex problems such as combinatorial optimization problems.

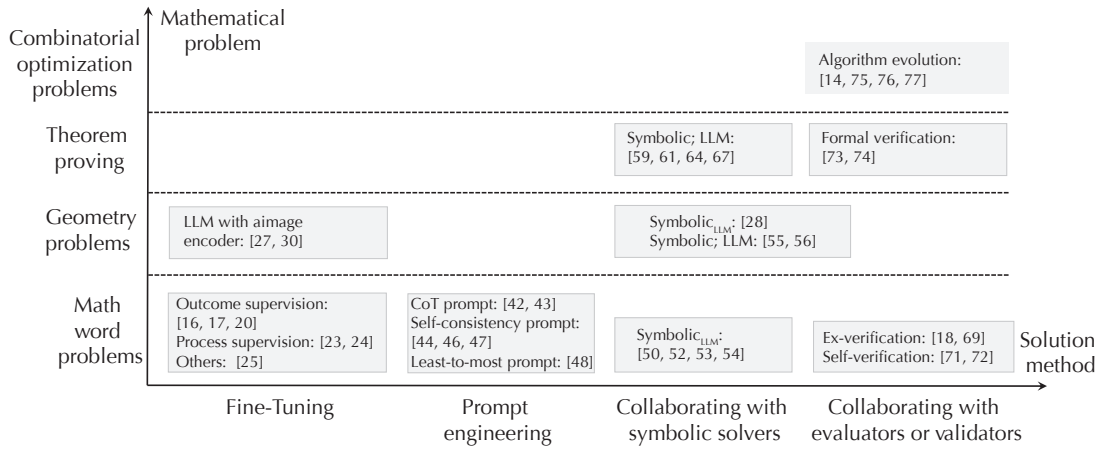


Figure 1. Classification System of SMP-LLM.

We employed the chi-square test to verify the correlation between the solution methods and mathematical problem types. The analysis was based on the citation counts of various mathematical problems under different solution methods, as presented in Figure 1. The null hypothesis proposed is that the solution methods and mathematical problem types are independent, with a significance level set at $\alpha = 0.05$. The results of the test indicated a significant correlation between the two. (Pearson's chi-square statistic $\chi^2 = 22.8$ exceeded the critical value $\chi^2_{0.95}(9) = 16.92$)

3. FINE-TUNING

The fine-tuning method assumes that adjusting model parameters based on relevant datasets alone can enable LLMs to solve simple mathematical problems. These mathematical problems mainly include math word problems and simple geometry problems with few reasoning steps.

3.1 Math Word Problems

Existing researches mainly focus on increasing the scale and diversity of datasets to improve the performance of the fine-tuned LLMs. According to whether the data used for fine-tuning are outcome data or process data, existing researches can be subdivided into outcome supervision and process supervision. Outcome supervision provides feedback on the final results, process supervision provides feedback for each intermediate reasoning step [15]. Works on outcome supervision include:

Yu et al. employed three methods to rephrase math word problems and constructed the MetaMathQA dataset for fine-tuning LLMs [16]. The first method involves reformulating math word problems using LLMs. The second is reverse reasoning, where known numerical terms in the problem will be replaced with variables, and the problem is rewrite to deduce the variables based on the answer. The third method supplements the second method by appending the statement *If we know the answer to the above problem is* a_i^**, what is the value of the unknown variable x?* directly behind the math word problem after replacing the variables. Yu et al. fine-tuned LLaMA-2 on MetaMathQA, creating the MetaMath language model for mathematical reasoning.

Xu et al. proposed a two-stage fine-tuning method for injecting diverse symbolic knowledge into LLMs [17], believing that the inherent correlations among various symbolic knowledge can enhance the performance of LLMs on symbolic tasks, including math word problem. In the injection stage, the authors emphasize using the intrinsic connections between different symbols in tasks such as mathematical reasoning, code generation, and knowledge graph construction to comprehensively learn various symbolic knowledge. In the infusion stage, both symbolic data and general instruction fine-tuning were utilized to balance the model's symbolic and language-related abilities. Specifically, symbolic knowledge related to mathematical reasoning mainly comes from math word problem datasets such as GSM8k [18], MATH [1], and AQUA [19].

Inspired by the way students learn to correct their mistakes, An et al. proposed a method called LEMA that uses a new type of data pairs called *error-correction* to fine-tune LLMs [20]. In the data pair generation phase, LEMA uses multiple LLMs like LLaMA and GPT series to collect correct reasoning paths. Then, using GPT-4 [21] as the *corrector*, LEMA generates *error-correction* data pairs through four steps: identifying errors, explaining the reasons for errors, correcting errors, and generating the final answers. In the LLMs fine-tuning phase, LEMA utilizes QLoRA [22] to fine-tune LLMs used for solving mathematical problems on reasoning paths and *error-correction* data pairs. The bottleneck of LEMA's performance lies in the ability of the *corrector*, for GPT-4 finds it difficult to correct challenging problems.

Due to process supervision is usually superior to outcome supervision [15], certain research endeavors utilize data from the reasoning process for fine-tuning. Yuan et al. enhanced the fine-tuning dataset [23] by generating reasoning paths. Specifically, they utilized the zero-shot Chain-of-Thought (CoT) capability of LLMs to generate reasoning paths and filtered out erroneous paths and other reasoning paths with identical equation lists.

Luo et al. proposed an evolutionary instruction feedback reinforcement learning method WizardMath [24] to enhance the mathematical problem-solving ability of LLaMA-2. WizardMath first generates diversified mathematical instruction data automaticlly using the math-specific Evol-Instruct. Then, trains the Instruction Reward Models (IRM) and Process supervision Reward Models (PRM), where the IRM assesses the quality of evolutionary instructions and the PRM evaluates each solution step. Finally, it conducts Proximal Policy Optimization (PPO) training based on the evolutionary instructions obtained from GSM8k [18] and MATH [1] and the instruction rewards and answer rewards obtained from IRM and PRM.

Few research works focus on the computational cost of fine-tuning. Full-model fine-tuning (FFT) requires significant computational resources. An approach called parameter efficient fine-tuning (PEFT) involves fine-tuning only a small portion of the external parameters, rather than the entire backbone network model, which has shown success on pre-trained models. Hu et al. further investigated the optimal settings and effects of PEFT on open-source LLMs [25], focusing on math word problems and common sense reasoning as downstream tasks. The conclusions are as follows. (1) The optimal positions for series adapter, parallel adapter, and Low-Rank Adaptation (LoRA) [25] are respectively: after the MLP layer, parallel to the MLP layer and both after the attention layer and MLP layer. (2) The open-source language model LLaMA-13B [26] using the PEFT approach outperforms GPT-3.5 in solving math word problems.

Since math word problems are targeted at elementary school students, solving such problems mainly relies on text comprehension ability, demanding lower logical reasoning and numerical calculation skills compared to other types of math problems. Therefore, most math word problems can be addressed by fine-tuning LLMs.

3.2 Geometry Problems

LLMs cannot directly process geometric figures, and current Multimodal Large Language Models (MLLMs) also struggle to accurately understand basic geometric elements and their relationships, these models have limitations in solving mathematical problems with geometric figures. Therefore, researchers combine LLMs with image encoders to solve geometry problems. Typical works include:

Gao et al. constructed a multimodal geometry dataset, Geo170 K [27], based on the Geometry3k [28] dataset. The process involved: first, convert manually annotated logical forms into detailed information items and geometric summaries, then generate QA pairs based on the information items and summaries. Using Geo170 K, four strategies were employed to construct the instruction fine-tuning dataset: Equation Solving (ES), Value Scaling (VS), Re-Formulating Condition as Unknown (RCU), Sentence Paraphrase (SP). G-LLaVA 7B and 13B models were obtained by fine-tuning on LLaVA [29], which consists of LLaMA-2 and a pretrained vision transformer.

Liang et al. proposed a multimodal mathematical reasoning model called UniMath [30]. To tackle geometry problems, math word problems, and table-based mathematical reasoning problems, UniMath fine-tuned a T5 model [31] and added a Vector Quantized Variational Autoencoders (VQ-VAE) [32-33]

as an image encoder that's specifically designed for geometry problems. VQ-VAE transforms geometry image patches into new tokens, and concatenates them with textual tokens to make the input. Through joint training across tasks on datasets SVAMP [34], GeoQA [35], and TableMWP [36], and fine-tuning on MathQA [37] and UniGeo-Proving [38] datasets, UniMath enhances mathematical reasoning abilities including geometry problem solving.

The above methods work well for geometry problems with simple shapes and few reasoning steps (usually no more than 5 steps). However, for complex geometry problems such as those in Olympic competitions, it is still challenging to address them effectively through fine-tuning LLMs.

3.3 Other Fine-tuning Based Research Work

There are also a few works that do not target specific types of mathematical problems. For example, building upon previous work [39], Zhao et al. further proposed JiuZhang 2.0 [40], a unified Chinese language model specifically designed for multitask mathematical problem-solving. First, the authors devised an architecture based on Mixture-of-Experts (MoE) to transfer and share mathematical knowledge across different tasks, thereby enhancing the performance of each task. Second, they designed a multitask continual pretraining and fine-tuning strategy, including masked token prediction, mathematical logic recovering, and solution checking, where masked token prediction serves as a common objective for LLMs and PLMs. The aim of mathematical logic recovering is to improve the model's understanding of mathematical logic, and solution checking aims to enhance the model's ability to identify and correct errors in the output. Third, to further enhance the model's general ability to solve various complex tasks, the authors utilized LLMs as a complementary model to refine the generated solutions.

Summary: Fine-tuning methods can improve the performance of LLMs in solving math word problems and simple geometry problems. However, there are two main limitations: first, it requires sufficient annotated data for effective training, leading to a high dependency on data. Second, the inherent weakness in reasoning ability of autoregressive LLMs cannot be significantly improved through fine-tuning alone, making this method inadequate for addressing mathematical problems involving complex reasoning.

4. PROMPT ENGINEERING

By providing specific inputs, prompt engineering can assist models to better understand the requirements of downstream tasks, reduce the gap between the model and the tasks, and enhance the model's performance and generalization ability on specific tasks.

This method is mainly used to solve math word problems. Typical works can be summarized into three categories: Chain-of-Thought prompting (CoT), self-consistency prompting and least-to-most prompting.

First, the primary technique employed is the CoT prompting method [41]. The CoT prompting technique induces models to mimic the logical chains of human thoughts to solve reasoning tasks that require multiple steps. Typical works include:

Using the *Let's think step by step* prompt, LLMs can generate reasoning chains step by step for the examples provided, but the generated chains often contain errors. To mitigate the impact of erroneous reasoning chains, Zhang et al. proposed an automatic CoT prompt method called Auto-CoT [42], which clusters problems into multiple clusters and implements diverse sampling based on these clusters to automatically generate examples for CoT.

Huang et al. proposed a method called CoT-Influx [43] to enhance the mathematical reasoning capability of LLMs using CoT. This method employs a coarse-to-fine pruner, which first identifies as many key CoT examples as possible and then prunes out unimportant tokens within the context window, so as to address the challenge of example selection caused by the limited context window length.

Second, self-consistency prompting. In this method, the LLMs will first generate several prompts, then select the optimal prompt by voting. Typical works include:

Fu et al. indicate that prompts with higher reasoning complexity, which means CoTs with more reasoning steps, perform significantly better in multi-step reasoning than simple prompts [44]. In light of this, the authors propose complexity-based prompts, a simple and effective method for selecting multi-step reasoning examples. This method selects the top 8 training samples with the most reasoning steps from datasets such as GSM8K [18] and MathQA [37] as CoT prompts. These prompts are then used to generate multiple reasoning chains with GPT-3 175B [45], then simple reasoning chains are excluded, and answers are generated from the remaining complex reasoning chains using a majority vote approach.

Students may validate their problem-solving processes using different methods when solving math problems. Inspired by this, Imani et al. proposed the Math-Prompter method to enhance the reasoning ability of LLMs in solving math word problems [46]. First, replace the numerical terms in the problem with variables. Then, use Zero-shot CoT to generate different solutions in the form of algebraic expressions or Python functions. Third, evaluate the solutions by replacing input variables with random values to assess if there is a consensus among different solutions. If there is no clear consensus, repeat the process.

Also inspired by the idea mentioned above, Wang et al. propose a self-consistency strategy to replace the greedy strategy in CoT [47]. This strategy samples different reasoning paths and then selects the most consistent answer by marginalizing the sampled paths.

Last, least-to-most prompting. CoT prompts often perform poorly when meeting problems that are harder than the examples provided by the prompts. To overcome this generalization challenge from easy tasks to hard tasks, Zhou et al. proposed a least-to-most prompting strategy [48]. This strategy decomposes complex problems into two steps: initially querying the LLM to decompose the problem into sub-problems; then query the LLM to solve the sub-problems sequentially, in this step, the answers previously solved from the sub-problems contribute to solving each subsequent sub-problem.

Summary: Prompt engineering can guide LLMs to generate desired outputs without updating model parameters. However, it also has two limitations: firstly, manually designed prompts introduces human labor costs and biases, leading to incorrect outputs. Secondly, relying on CoT alone cannot significantly

enhance reasoning abilities. Similar to fine-tuning methods, prompt engineering cannot address mathematical problems involving complex reasoning.

5. COLLABORATING WITH SYMBOLIC SOLVERS

LLMs have rich prior knowledge, strong generalization and high flexibility, but weak reasoning ability and poor interpretability. Symbolic systems exhibit properties like composability, interpretability, and support for higher-order reasoning and multi-step inference, but also suffer from issues like combinatorial explosion and sensitivity to noise. Combining the two of them can provide a best-of-both-worlds situation, and has become a promising direction for solving mathematical reasoning problems. Referencing the neuro-symbolic architectures classification system proposed in [49], we categorize the paradigms of the collaboration between LLMs and symbolic solvers into two types:

(1) Symbolic$_{\text{LLM}}$. In this paradigm, LLMs do not perform the reasoning process, but only convert the problem into symbolic data, and then process the data by a symbolic solver.

(2) Symbolic; LLM. In this paradigm, LLMs and symbolic solvers iteratively interact as independent modules, each accomplishing the subtasks they are skilled at.

5.1 Math Word Problems

Solving math word problems primarily adopts the Symbolic$_{\text{LLM}}$ paradigm. Specifically, it involves using LLMs to translate natural language formatted mathematical problems into formal languages that symbolic solvers can process, such as code. Typical works include:

Chen proposed Program-of-Thoughts (PoT) to solve math word problems [50]. PoT utilizes LLMs such as Codex [51] to generate Python programs where Codex is a descendant of GPT-3, its training data contains both natural language and billions of lines of source code from publicly available sources. The generated programs are then executed by a Python interpreter to produce answers, thus separating complex computations from reasoning and language understanding.

Gao et al. proposed the Program-Assisted Language model (PAL) [52], which employs Codex [51] to parse natural language problems and generate programs as intermediate reasoning steps, while the solving process is delegated to interpreters like Python.

Yamauchi et al. proposed a mathematical reasoning framework called LPML [53] which combines CoT with Python REPL (Read-Eval-Print Loop). REPL is a standard library in Python, it provides a basic interactive environment for inputting and executing Python code line by line and displays results instantly. Unlike directly generating Python code to solve problems, LPML establishes an interaction between LLM and REPL, allowing LLM to simultaneously generate CoT and Python code. The code execution results are then fed back to LLM to induce corrections to the errors in CoT. LPML uses the consistency between CoT and Python code execution to enhance mathematical reasoning performance.

The above methods are limited by LLMs' ability of code generation. When facing problems with multiple processes, it is very difficult to ensure that LLM generates the correct code for each step.

Though programs can directly represent the solving process, some more complex math word problems require more abstract mathematical statements. To address this, He et al. proposed a method that combines LLM with an external symbolic solver [54]. LLM can translate math word problems into a set of mathematical statements composed of variables and equations (e.g., *Let b be how many apples she had in the morning after eating 2 apples [var b]. We have [eq b = a − 2]*), while the external symbolic solver is used to solve the equations in the mathematical statements.

5.2 Geometry Problems

Researches on solving geometry problems by combining LLMs with symbolic solvers is currently less, but it still encompasses the two paradigms mentioned above.

Lu et al. adopted the **Symbolic$_{\text{LLM}}$** paradigm. Specifically, the authors defined a geometric formal language consisting of predicate, literal, and primitives and proposed a geometric solving method based on geometric formal language and symbolic reasoning, called the Interpretable Geometry Problem Solver (InterGPS) [28]. InterGPS automatically parses the problem text and diagrams into formal language using rule-based text parsing and object detection. Then, it applies theorems as conditional rules, and performs symbolic reasoning step by step. To achieve more efficient and rational search paths, the authors designed a theorem predictor to provide theorem usage sequences for the symbolic solver.

Wu et al. [55] also adopted the **Symbolic$_{\text{LLM}}$** paradigm. The authors considered two limitations in existing methods: first, poor interpretability; second, the small scale and incomplete annotations of existing datasets make it hard for LLMs to comprehend geometric knowledge. To address these issues, the authors proposed the method called Explainable Geometry Problem Solving (E-GPS) [55]. E-GPS first parses geometric graphs and problem text into a unified formal language representation. Then, it uses a Top-Down Problem Solver (TD-PS) to obtain the answers the interpretable reasoning steps. To mitigate the data issues, the authors designed a Bottom-Up Problem Generator (BU-PG) to expand the dataset with various well-annotated constructed geometry problems.

For complex geometry problems, the cost of converting human proofs into machine-verifiable formats is very high, which leads to a severe scarcity of training data, current machine learning methods are not suitable for solving complex geometry problems. In response, Trinh et al. adopted the **Symbolic; LLM** paradigm and developed a Euclidean plane geometry tailored theorem-proving program: AlphaGeometry [56], in which LLM is responsible for generating auxiliary lines, and the symbolic system handles reasoning and computation. Specifically, the authors iteratively invoke three modules: deductive database (DD) [57], algebraic rules (AR) [56], and random auxiliary point addition, to automatically generate synthetic data in the form of <*premises*> <*conclusion*> <*proof*> triplets, thus avoiding manual annotation. Then use the generated one hundred million synthetic data to train an LLM capable of adding auxiliary points for geometric problems. In this process, DD and AR respectively performed symbolic reasoning and

operations, the former applies in inference rules on given premises repeatedly to derive new conclusions, while the latter handles addition and proportion relationships between the angles and edges. In geometric problem-solving, AlphaGeometry alternately called DD + AR + LLM to iteratively execute geometric reasoning and auxiliary point construction.

Using LLMs to solve or generate geometric problems is a potential research direction. Currently, a limitation of such methods is their relatively poor understanding of geometric figures. For example, InterGPS [28] can only parse relatively simple geometric shapes. existing methods still struggle to apply complex geometric figures, such as those in the International Mathematical Olympiad level problems. For instance, AlphaGeometry [56] directly conducts reasoning using formal representations, bypassing the process of converting figures into formal representations.

5.3 Theorom Proving

Currently, there have been some studies that combine LLMs with symbolic systems for theorem proving, mainly adopting the **Symbolic; LLM** paradigm. In these approaches, LLMs are mainly used to address bottleneck issues in theorem proving, namely, how to select appropriate premises from a large premise pool [58]. Typical works include:

Yang et al. developed an open-source Lean experimentation environment called LeanDojo [59], consisting of toolkits, data, models, and benchmarks. The data extracted from Lean in LeanDojo includes fine-grained annotation of the premises in the proofs. Utilizing the data, Yang et al. developed an LLM based retrieval-augmented prover: ReProver, using the retrieval-enhancement ability one can retrieve small sets of premises from mathlib [60], the mathematical library of Lean. To enhance the retrieval performance, hard negative instances need to be provided during training, which are negative instances difficult to distinguish. To address this, the authors proposed a simple strategy, which is to select negated premises from the same Lean source file that the true premises are defined in as negative instances.

Thakur and colleagues proposed a formal theorem proving method based on LLMs called COPRA [61]. COPRA employs a black-box LLM (GPT-4 [21]) as part of the stateful backtracking search strategy. During the search process, the strategy can select proof tactics and retrieve lemmas and definitions from an external database. Each selected proof tactic is executed within the theorem proving environment Coq [62] or Lean [63], and the execution feedback is used to establish prompts for the next strategy invocation.

Automatic formalization is the process of automatically converting natural language mathematical expressions into formal specifications and proofs, which is a crucial step in theorem proving. Research by Wu et al. [64] shows that using different scales of LLMs such as PaLM [65] and Codex [51], with a simple prompt *Translate the natural language version to an Isabelle version*, can correctly translate a considerable portion (25.3%) of mathematical competition problems into formal specifications in Isabelle/HOL [66]. Furthermore, experiments also demonstrate that automatic formalization can provide useful training data for neural theorem provers.

Welleck and Saha developed a proof step recommendation tool, LLMSTEP [67], which integrates LLM and Lean [63]. LLMSTEP sends the user's proof state to Pythia 2.8B [68], the LLM is fine-tuned on samples from Lean Mathlib [60]. Pythia then generates suggestions for proof steps. LLMSTEP employs Lean to verify the validity and completion of each suggestion.

The performance of the aforementioned work in theorem proving still has significant limitations, primarily due to the deficiency of LLMs in selecting premises. This not only requires LLMs to possess sufficient mathematical knowledge but also demands LLMs to search for suitable premises from a vast candidate space.

Summary: LLMs are skilled at understanding natural language mathematical descriptions, identifying problem types, and generating problem-solving strategies, they are capable of transforming complex textual expressions into mathematical concepts and relationships. Symbolic solvers, on the other hand, demonstrate excellence in precise calculations, logical reasoning, and formal verification, ensuring the accuracy of computations and the rigor of reasoning. The collaboration between LLMs and symbolic solvers can effectively solve complex mathematical problems described in natural language and involving symbolic operations and exact calculations.

However, they still show significant limitations on the following types of mathematical problems: First, cross-media problems involving abstract graphics. Neither symbolic solvers nor current multimodal LLMs can achieve precise parsing and spatial relationship reasoning of abstract geometric images. Second, problems that are highly abstract and lack clear patterns. Such problems often lack recognizable standard patterns or require extremely strong creative reasoning, such problems include: certain pure mathematical proving and frontier theoretical mathematical issues. Third, problems relying on large-scale numerical simulations. For Monte Carlo simulation problems, the collaborative advantage of symbolic solvers and LLMs is not evident; these problems rely more on specialized numerical computation tools.

6. COLLABORATING WITH EVALUATORS/VALIDATORS

Many mathematical problems possess the characteristic of *hard to solve, easy to evaluate*, these problems can be solved by collaborating LLMs with evaluators/validators, essentially exploration-exploitation approaches. During the exploration phase, LLMs generate candidate complete solutions or individual solution steps. In the exploitation phase, evaluators/validators discriminate or select among the candidate solutions or steps. This process is usually executed iteratively to generate the optimal solution.

6.1 Math Word Problems

For math word problems, some research works directly utilize LLMs to generate solutions, corresponding verification methods include external-verification and self-verification. In external-verification, independent modules are used to evaluate the generated solutions of the LLM, and in self-verification, the same LLM acts as the verifier to provide feedback on the generated solutions. Works on external-verification include:

Cobbe et al. constructed a dataset of 8.5K high-quality math word problems, GSM8K [18], which has become an important benchmark for evaluating the mathematical reasoning abilities of LLMs. None of the GPT-3 6B/175B was able to achieve satisfactory performance on this dataset. To address this, the authors trained an evaluator to estimate the correctness of the model-generated solutions, the correctness of the solution is determined solely by whether it produces the correct answer. When testing, the evaluator will select the highest scored solution from the candidate solutions as the final solution.

Li et al. proposed a reasoning step diversification validator, DIVERSE, to further enhance the reasoning ability of LLMs [69]. The formal validator first generates diverse prompts to explore different reasoning paths for the same problem. Second, composite voting and verification strategies to filter out incorrect answers through weighted voting. Third, it verifies each reasoning step individually rather than the entire reasoning chain, taking full advantage of the correct steps within incorrect reasoning. DIVERSE is fine-tuned on deberta-v3-large [70] and uses OpenAI's davinci series models for reasoning, it achieved state-of-the-art results on tasks such as math word problems.

Some research works take advantage of code verification to generate solution schemes in the form of code and use self-verification to evaluate the outcomes. Typical works include:

Madaan et al. proposed Self-Refine [71] method uses LLMs to generate initial outputs, the same LLM is then employed as a validator to provide feedback to iteratively refine the generated outputs. In this process, the LLM simultaneously acts as the generator, validator, and improver, thus eliminating the need for any supervised training data, additional training, or reinforcement learning. For math word problems, Madaan et al. used PAL [52] to generate solutions in Python. By utilizing Self-Refine, the model can identify errors in the code and improve the solutions through an iterative process of introspection and feedback.

Building on the work [52], Zhou et al. further proposed the explicit code-based self-verification (CSV) [72]. CSV employs the zero-shot prompt with GPT4-Code, encouraging GPT-4 [21] to generate code to self-verify its answers. If the verification status is *incorrect*, the model will automatically correct the solution, this is similar to how humans rectify errors during a math exam.

The above methods draw inspiration from how humans self-correct errors during math exams, thereby enhancing LLMs' ability in solving math word problems. But they also introduce certain drawbacks, such as the increase of model complexity and higher training costs. Self-verification is more efficient and convenient compared to external-verification because it uses the same LLM as the verifier so it can gain feedbacks quickly. However, it may be limited by the LLM's own reasoning ability and fail to provide a completely objective evaluation result. At the same time, though external-verification may provide a more objective and accurate evaluation result, it increases the model's complexity and computational expenses because of the need to introduce additional modules.

6.2 Theorem Proving

In the field of automated theorem proving, some research efforts combine validators such as Metamath [16] and Isabelle [66] with LLMs to enhance theorem proving performance. Typical works include:

Polu and Sutskever applied transformer-based LLMs to automated theorem proving and proposed an automatic theorem prover and proof assistant for Metamath formal language: GPT-f [73]. GPT-f uses a trained decode-only transformer architecture LLM to generate the proving process, and uses a Metamath verifier to verify it. To enhance the proof search efficiency of LLMs, the authors annotated the LLM generation process using the verifier, and iteratively trained the value function using the annotated data to improve the performance of GPT-f. GPT-f also discovers several new short proofs, 23 short proofs have been included to the Metamath library.

Different from existing methods that predict one proof step at a time, First et al. proposed the theorem proving approach Baldur [74], which is a theorem proving method based on LLMs trained on natural language texts and codes and fine-tuned on proofs. It generates the entire proof of the theorem at one time rather than predicting one step at a time. Specifically, Baldur consists of three modules: the proof generation module, which is an LLM that generates candidate proofs based on the input theorem statement; Isabelle, a higher-order logic (HOL) theorem prover [66], it is used to verify the candidate proofs, if the verification fails, it will return an error message; the proof correction module, also an LLM, it corrects the proof process based on the error message returned, then passes it to Isabelle to verify, iterates this process until Isabelle passes the verification.

Collaboration between LLMs and evaluators/validators can effectively demonstrate strong theorem proving capabilities, but it relies on the scoring signals generated by the evaluators/validators. For multi-step proof processes, it is difficult to provide rich scoring signals for each inference step for proof process correction.

6.3 Combinatorial Optimization Problems

Combinatorial optimization problems involve finding the optimal solution from a finite solution space. In addition to the objective function that needs to be maximized or minimized, combinatorial optimization problems are also accompanied by a series of constraints that solutions must satisfy. The solution space of combinatorial optimization problems is typically large and discrete; thus, researchers proposed the Algorithm Evolution framework [75] shown in Figure 2 to manage these problems. It involves prompting the LLM to generate initial solution algorithms in code form, and continuously evolving these algorithms by maintaining an algorithm pool. The basic process is: prompt LLM to execute operations such as crossover, mutation, and creation on algorithms selected from the algorithm pool, and use the evaluator to select algorithms to add to the algorithm pool, iterate this process until an optimized solution is found.

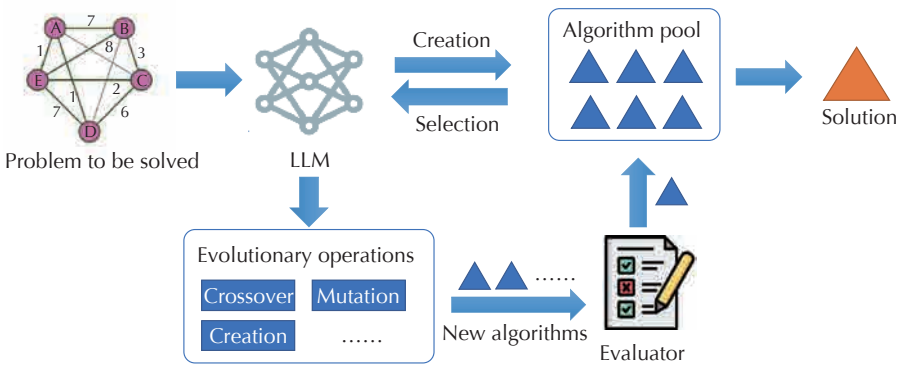


Figure 2. Algorithm Evolution Framework.

Typical works under the framework include:

Liu et al. proposed the AEL method [75] that can automatically generate optimization algorithms using LLMs under the evolutionary framework. AEL achieves algorithm construction and evolution by prompting LLMs to execute operations such as initialization, crossover, and mutation. This process eliminates the need for model training, significantly reducing human workload and domain knowledge requirements. Furthermore, Liu et al. utilized the AEL framework to design the Guided Local Search (GLS) algorithm [76]. GLS employs a utility function to guide lower-level local search algorithms, which can effectively explore the search space and prevent the algorithm from falling into a local optimal state. GLS was applied to solve the Traveling Salesman Problem (TSP). AEL can automatically evolve excellent GLS algorithms within two days with minimal human intervention and no need for model training.

Romera-Paredes et al. proposed a combinatorial optimization problem solving method that combines LLMs with evaluators named FunSearch [14]. Though FunSearch and the research by Liu et al. [75-76] are independent of each other, they share similar ideas, so it can still be classified under the category of algorithm evolution. In each iteration, FunSearch selects the highest scoring program from the program library to generate prompts and inputs them into the LLM to generate new programs. If the new program passes evaluation of the evaluator, it will be stored in the program library, the solution to the problem is obtained by retrieving the highest scored program of the program library.

Liu et al. utilized LLMs to solve combinatorial optimization problems and proposed the LLM-driven Evolutionary algorithm(LMEA) method [77]. In each generation of evolutionary search, LMEA constructs prompts to guide the LLM in selecting parent solutions from the current population and performs crossover and mutation to generate offspring solutions. LMEA evaluates and selects new solutions to be included in the next generation population. LMEA employs a simple self-adaptive mechanism, it balances exploration and exploitation by controlling the temperature of the LLM to prevent the search from falling into a locally optimal solution. LMEA can quickly adapt to different optimization problems due to the fact that optimization problems can be described in natural language descriptions and LLMs can be guided by the desired solution properties.

With the algorithmic evolution framework, LLMs and evaluators/validators can collaborate to solve some complex combinatorial optimization problems, but they have a high dependency on scoring signals. For certain combinatorial optimization problems, it is challenging to generate rich scoring signals for algorithmic evolution.

6.4 Reinforcement Learning-Driven Mathematical Problem Solving

The LLMs used in the aforementioned studies are usually autoregressive models formed through self-supervised learning or supervised fine-tuning. Recently, a series of LLMs that incorporate reinforcement learning, also known as reasoning models, have emerged, such as OpenAI's o1/o3 and DeepSeek-R1 [78], they show significant improvement in complex reasoning tasks, including mathematical problem-solving. The main reason for this improvement is that, compared to supervised learning, reinforcement learning can better address the two key characters of complex mathematical problem-solving. First is delayed rewards, the effect of each reasoning step may only become apparent after numerous subsequent steps, and intermediate rewards may be sparse or even zero. Reinforcement learning can effectively handle the delayed reward issue through value functions/networks and policy optimization. Second is balancing exploration and exploitation, Supervised learning can only leverage existing labeled data, and lacks exploration mechanisms, but reinforcement learning can explore new strategies to discover potentially better solutions, avoiding local optima or ineffective reasoning paths.

Such works can also be classified under the "Generator-Evaluator" paradigm, where the generator is represented by the policy, responsible for generating steps or strategies for problem-solving; the evaluator is represented by the value function or reward model, it provides feedback by assessing the quality of the generated action a. Among these works, the models primarily focused on mathematical problem-solving tasks include: WizardMath [24], DeepSeekMath [79], and R^3 [80].

Luo et al. proposed WizardMath [24], which enhances the mathematical reasoning ability of LLMs through Reinforcement Learning from Evol-Instruct Feedback (RLEIF). This approach combines Math Evol-Instruct and process supervision. It generates diverse mathematical instruction data through both downward and upward evolution.It introduces Instruction Quality Scoring and Process Supervision to optimize the model.

Shao et al. proposed DeepSeekMath [79], an LLM which focuses on mathematical reasoning. The training process of the model is: First, pre-train DeepSeek-Coder-Base-v1.5 7B on 120 billion high-quality math-related data. Second, proceed reinforcement learning optimization by Group Relative Policy Optimization (GRPO), GRPO significantly reduces the need for training resources by using the average reward of multiple sampled outputs as baseline, and significantly reduces the need for training resources, avoiding the need for value networks as required in Proximal Policy Optimization (PPO).

Xi et al. proposed a method named R^3 [80], it can enhance the reasoning ability of LLMs through Reverse Curriculum Reinforcement Learning. R^3 employs the outcome supervision as the evaluator, achieving an effect similar to Process Supervision while avoiding the high cost of manual annotation.

Specifically, R^3 starts from the intermediate states of the correct demonstrations and gradually moves backward to provide approximated Process Supervision signal.

Research on reinforcement learning-based reasoning models is still in its early stages. Employing reasoning models for mathematical problem-solving still faces two major challenges: First, the issue of delayed rewards leads to the temporal credit assignment problem, which refers to how to properly allocate delayed rewards to reasoning steps. Second, in the tradeoff of exploration and exploitation, excessive exploration or exploitation may result in underthinking and overthinking. The former refers to the model's tendency to frequently switch between reasoning strategies without processing deep inference on any single strategy. The latter refers to the model's lack of exploration, meaning that the model may process excessive reasoning steps within a particular strategy, leading to either failure in problem-solving or the generation of redundant reasoning paths.

Summary: In mathematical problem solving, the introduction of evaluators/validators to LLMs demonstrates significant advantages. The system can detect errors in real time at each reasoning step and trigger correction mechanisms, effectively preventing error accumulation and amplification. This approach can further enhance the exploratory capabilities in reasoning after integrating with reinforcement learning frameworks, achieving or even surpassing human-level performance on challenging mathematical problems such as theorem proving and combinatorial optimization.

However, this collaborative mechanism also faces three core limitations. First is the sparse reward problem: during long-chain reasoning processes (e.g., the average proof length for olympiad theorems is 50 steps [56]), the evaluator often fails to provide dense and timely scoring signals, impacting reasoning efficiency. Second is the exploration suppression issue: due to the inherently conservative tendency of evaluators to identify and penalize errors, it may restrict the model's exploration of novel reasoning paths, even leading to overthinking. Last is the computational burden problem: each reasoning step requires additional evaluation, significantly increasing computational costs and response latency, which could become a critical bottleneck in practical deployments.

7. DATASETS

Currently, several mathematical problem datasets have been constructed specifically for performance testing or model training. Figure 3 categorizes different datasets according to three problem types, the arrows indicate that the subsequent dataset introduced some or all of the mathematical problems from previous datasets.

The experimental results on the main benchmarks from the key references are provided in Appendix A.

Table 2(a) ~ 5 provide some brief descriptions of these datasets, along with typical examples of the problems.

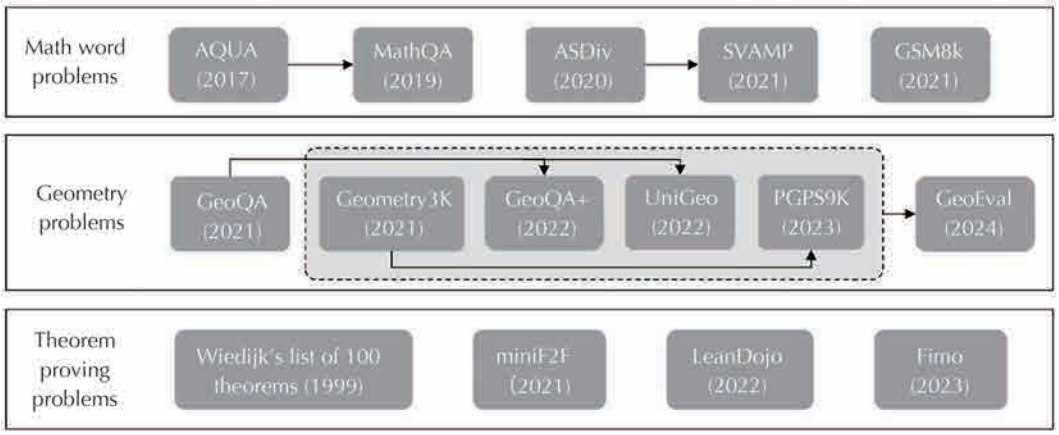


Figure 3. Mathematical Problem Datasets.

Table 2(a). Math Word Problem Datasets.

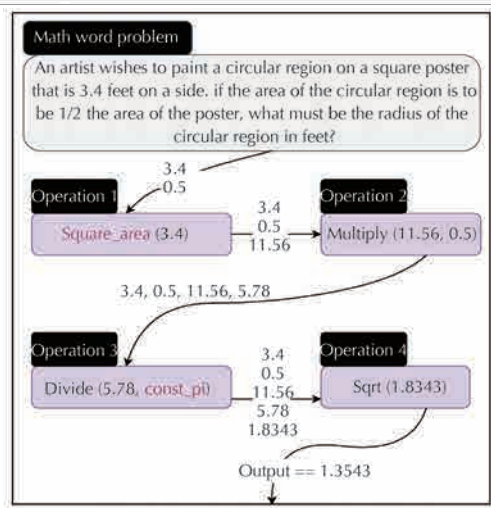
Dataset	Brief Descriptions	Problem Examples
AQUA [19] (2017)	AQUA contains 100,000 problems constructed through crowdsourcing, the problems are based on 34,202 seed problems from GMAT and GRE exams. Each problem is decomposed into four parts: the problem description, options, the rationale, and the correct option.	**Problem 1:** **Question:** Two trains running in opposite directions cross a man standing on the platform in 27 seconds and 17 seconds respectively and they cross each other in 23 seconds. The ratio of their speeds is: **Options:** A) 3/7 B) 3/2 C) 3/88 D) 3/8 E) 2/2 **Rationale:** Let the speeds of the two trains be x m/sec and y m/sec respectively. Then, length of the first train = 27x meters, and length of the second train = 17 y meters. (27x + 17y) / (x + y) = 23 → 27x + 17y = 23x + 23y → 4x = 6y → x/y = 3/2. **Correct Option:** B
MathQA [37] (2019)	MathQA contains 37,200 problems from AQUA [19], along with corresponding multiple-choice options and solution processes. The authors developed a specialized annotation platform and used crowdsourcing to accurately annotate the solution processes. In addition to mathematical reasoning performance evaluation, this dataset can also be used to fine-tune or prompt LLMs to improve reasoning performance and interpretability.	

Table 2(b). Math Word Problem Datasets.

Dataset	Brief Descriptions	Problem Examples
ASDiv [81] (2021)	ASDiv (Academia Sinica Diverse) contains 2,305 math word problems, covering a wider range of text patterns and most problem types found in elementary school curricula. Each math word problem is labeled with the problem type and grade level (indicating difficulty). To assess the diversity of the problems, the authors proposed the lexicon usage diversity (LD) index based on BLEU [81] to evaluate the diversity of the dataset.	
SVAM [34] (2021)	SVAMP includes 2,305 math word problems, covering a broader range of text patterns and most problem types in elementary school mathematics. The problems are sourced from the arithmetic subset ASDiv-A of ASDiv [81] and have undergone three types of modifications. These modifications are aimed to evaluate the model's capabilities in three aspects: whether the answers depend on the problems themselves, whether slight changes in problem text affect the reasoning steps, and whether changes in the surface structure of the problem text affect the answers or reasoning steps.	
GSM8K [18] (2021)	GSM8K contains 8.5K high-quality, human-constructed elementary math problems, with 7.5K problems for training and 1K problems for testing. The solution processes primarily use basic arithmetic operations such as +, –, ×, ÷, and the steps to solve the problems range from 2 to 8. The solution processes are described in natural language rather than purely mathematical expressions to better evaluate the language model's ability to solve math word problems.	Dean's mother gave him $28 to go to the toy store. Dean bought 6 toy cars and 5 teddy bears. Each toy car cost $2 and each teddy bear cost $1. His mother then feels generous and decides to give him an extra $10. How much money does Dean have left? **Ground Truth:** The cost of the toy cars is 6 cars × $ 2/car = $ << 6*2 = 1 2>> 12. The cost of the teddy bears is 5 bears × $ 1/bear $ << 5*1 = 5 >> 5. The total cost of the toys is $12 + $5= $ << 12 + 5 = 17 >> 17. Adding the two amounts of money his mother gave him, we find that Dean has $28 + $10 = $ << 28 + 10 = 38 >> 38 to spend. Dean has $38 – $17= $21 left. A: 21

Table 3(a). Geometry Problem Datasets.

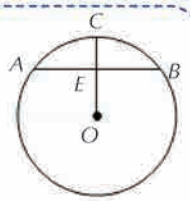
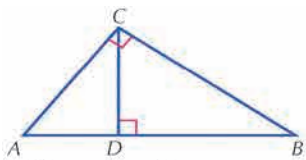
Dataset	Brief Descriptions	Problem Examples
GeoQA [35] (2021)	GeoQA contains 4,998 real geometry problems from Chinese middle school exams. Each problem can be represented as (*t, d, c, i, e, t, k, p*). *t* is the problem text; *d* is the geometric figure; $c = \{c_1, c_2, c_3, c_4\}$ are the answer options; *i* is the answer index, where $c_i \in c$; *e* is the natural language-based problem-solving explanation; *t* is the problem type, including *Angle Calculation*, *Length Calculation*, and other types; *k* is the related knowledge point, such as the Pythagorean Theorem; *p* is the Annotated Programs, represented in the language form consisting of operations (e.g., Add, Sin), constants (e.g., π, 30), problem variables, and process variables.	As shown in the figure, in ⊙O, AB is the chord, OC⊥AB, if the radius of ⊙O is 5 (N0) and CE=2 (N1), then the length of AB is () A.2 B.4 C.6 D.8 **Answer**: D.8  **Problem Type**: Length Calculation **Knowledge Points**: Vertical Diameter, Pythagorean Theorem **Problem Solving Explanations**: OE = OC-CE = 5-2 = 3. According to the Pythagorean Theorem, AE = $\sqrt{OA^2 - OE^2} = \sqrt{5^2 - 3^2} = 4$. Thus, AB = 2AE = 8. **Annotated programs:** Minus | N0 | N1 | PythagoreanMinus | N0 | V0 | Double | V1 Step1: Minus (N0, N1) = 5 – 2 = 3 (V0) Step2: PythagoreanMinus (N0, V0) = $\sqrt{5^2 - 3^2} = 4$ (V1) Step3: Double (V1) = 2 × 4 = 8 (V2)
Geometry-3K [28] (2021)	Geometry3K contains 3,002 geometry problems. Annotators labeled the problem text, geometric figures, options, and correct answers. The problem text and geometric figures are described using literals, each literal consists of a predicate and a set of arguments (constants, variables, or literals). For example, *AreaOf* (*Triangle* (*A, B, C*)) represents the area of Δ*ABC*. This geometry formal language bridges the semantic gap between text and geometric figures, facilitating symbolic problem solving.	 In triangle ABC, AD = 3 and BD = 14. Find CD. Choices: A. 6.0 B.6.5 C.7.0 D. 8.5 Answer: B **Diagram formal language** Triangle (A, B, C) Triangle (A, C, D) Triangle (B, C, D) PointLiesOnLine (D, Line (A, B)) Perpendicular (Line (A, C), Line (B, C)) Perpendicular (Line (C, D), Line (A, B)) **Text formal language** Triangle (A, B, C) Equals (LengthOf (Line (A, D)), 3) Equals (LengthOf (Line (B, D)), 14) Find (LengthOf (Line (C, D)))

Table 3(b). Geometry Problem Datasets.

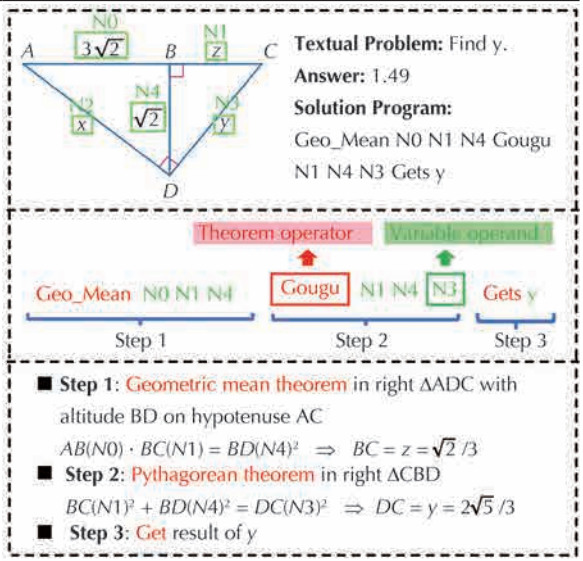
Dataset	Brief Descriptions	Problem Examples
GeoQA+ [82] (2022)	GeoQA+ annotates an additional 2,518 geometry problems on the basis of GeoQA [35], including 636 area-type problems, which are not presented in GeoQA. Compared to the average solving steps of 1.96 in GeoQA, the newly annotated problems are more challenging, with an average solving steps of 2.61.	
UniGeo [38] (2022)	UniGeo contains 4,998 geometric computation problems from GeoQA [35], as well as geometric proof problems from the IXL2 website (https://www.ixl.com/math/geometry). Each proofs problem is annotated with multi-step proofs including reasoning and mathematical expressions. The proof can be easily reformulated as a proving sequences that shares the same formats with the annotated program sequence for calculation problems.	
PGPS9K [83] (2023)	PGPS9K contains 9,022 geometry problems, with 2,891 problems selected from Geometry3K [28], and the rest from mathematics course websites. The problems in PGPS9K are categorized into 30 types, such as Line Segment, Angle, and Circumference and Area of Circle. Each problem's derivation steps are finely annotated, with each step consisting of an operator and several operands. PGPS9K defines 34 operators (such as geometric theorems) and 55 operands (including problem variables, process variables, unknown variables, and constants).	
GeoEval [84] (2024)	GeoEval's geometry problems come from datasets such as Geometry3K [28], PGPS9K [83], UniGeo [38], and GeoQA+ [82], and are divided into four subsets. (1) The GeoEval-2000 dataset contains 2,000 problems. (2) The GeoEval-backward dataset is created by the corresponding backward problems of 750 selected problems of the GeoEval-2000 subset, by providing answers o reverse-engineering the masked numbers. (3) The GeoEval-aug subset is formed by rephrasing 2,000 problems from the GeoEval-2000 subset using GPT-3.5, this can address the problem of data leakage. (4) The GeoEval-hard subset contains 300 geometry problems, it focuses particularly on solid geometry and analytic geometry, it can help enhance problem type diversity.	

Table 4. Theorem Proving Datasets.

Dataset	Brief Descriptions and Problem Examples (portion)
Wiedijk's list of 100 theorems [85] (1999)	In July 1999, at a mathematics conference, mathematicians Jack Abad and Paul Abad unveiled the list of *The Hundred Greatest Theorems*. Their ranking was based on the theorem's status in the literature, the quality of the proof, and the unexpectedness of the result. The top three theorems were: The Irrationality of the Square Root of 2, the Fundamental Theorem of Algebra, and the Denumerability of the Rational Numbers. In [85], the formalization of these theorems in systems such as Isabelle [66], Coq [62], and Lean [63] was documented.
miniF2F [86] (2021)	miniF2F is a dataset of manually formalized statements of Olympiad-type problems, aligned in Lean [63], Metamath [87], and Isabelle [66], providing a cross-platform benchmark for formal mathematical reasoning. The formalized statements in miniF2F come from multiple sources, covering exercises from high school and undergraduate levels to Olympiad problems, mainly from AIME (American Invitational Mathematics Examination), AMC (American Mathematics Competition), IMO problems, and informal datasets from MATH [1], allowing miniF2F to encompass a broader range of difficulties. MiniF2F also covers various subfields and proof strategies in mathematics, with a primary focus on algebra, number theory, and inequalities.
LeanDojo Benchmark [59] (2022)	LeanDojo is a benchmark for premise selection and theorem proving, including 98,734 theorems and proofs extracted from the Lean mathematics library [60], covering topics such as analysis, algebra, and geometry. Unlike the existing Lean dataset, the LeanDojo Benchmark also includes definitions of 130,262 premises, comprising not only theorems but other definitions usable as premises. Additionally, the dataset contains 217,776 tactics, with 129,243 tactics involving at least one premise. Among the tactics with premises, the average number of premises is 2.13.
Fimo [88] (2023)	Fimo is a dataset of human-verified auto-formalized statements of IMO-level mathematical problems aligned in Lean language. It comprises 149 formal problem statements, accompanied by both informal problem descriptions and their corresponding LATEX-based informal proofs.

Table 5. Theorem Proving Datasets.

Dataset	Brief Descriptions and Problem Examples (portion)
MATH [1] (2021)	Hendrycks et al. constructed the MATH dataset [1], which contains 12,500 mathematics competition problems sourced from AMC 10, AMC 12, AIME and other mathematics competitions. Unlike previous datasets, most problems in MATH cannot be solved directly using the K-12 mathematic tools but require skills and heuristic methods. The problems in the MATH dataset span different subjects and difficulty levels. The seven subjects are Prealgebra, Algebra, Number Theory, Counting and Probability, Geometry, Intermediate Algebra, and Precalculus. Each problem includes a multi-step solution and a final answer, and is assigned a difficulty rating from 1 to 5.

8. FUTURE RESEARCH DIRECTIONS

8.1 Geometric Figure Analysis

In courses such as plane geometry, solid geometry, and analytic geometry, there are numerous geometric figures composed of lines, rectangles, circles, etc. Parsing these geometric shapes to identify their elements and topological relationships, and representing them formally, is a prerequisite for collaborative reasoning with LLMs. Although AlphaGeometry [56] can solve IMO-level geometric problems, it directly uses the formal language of geometric figures as input.

In practice, parsing geometric figures into formal language is not easy for two main reasons: first, unlike natural images in the field of computer vision, geometric figures have sparse visual features, their color, texture, and background information are much less than those of natural images. Second, geometric figures exhibit a phenomenon different from natural images known as *isomorphism but different shapes*, meaning that geometrically different figures (as shown in Figure 4) may express the same topology.

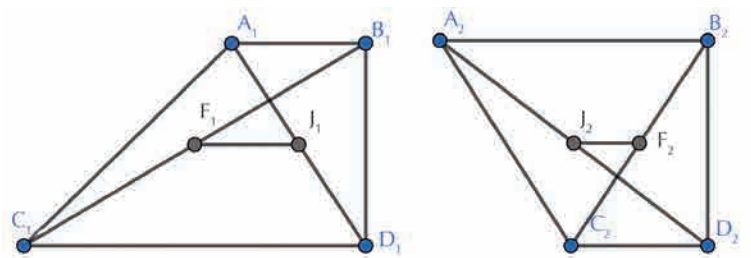


Figure 4. isomorphism but different shapes.

This makes it difficult to apply object detection models that perform well on natural images to geometric figure parsing. Our test results indicate that object detection models like classic Faster R-CNN [89] and popular models like DETR [90], CenterNet [91], etc., achieve an average accuracy of 85.3% on MSCOCO [92], but only 13.3% on GeoQA [35]. Additionally, we also employed GPT-4V to describe 50 randomly selected geometric figures from GeoQA [35], using the prompt *Please list out the known information that is directly reflected in the given geometric diagram*. Only 6 descriptions were completely correct, Figure 5 lists 4 typical failure cases.

8.2 The Comprehension and Generation of Mathematical Expressions

Mathematical problems and their solution processes usually include many mathematical expressions, there are three challenges in understanding and generating these expressions:

First, mathematical expressions contain specific symbols with specific meanings, such as $\int$, ∂, $\sum$, $\prod$. Understanding the meanings of these symbols is essential for accurately understanding mathematical expressions.

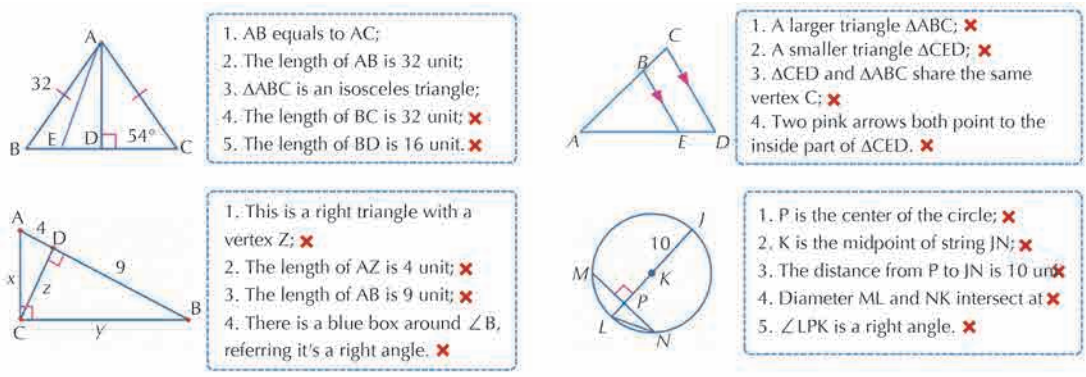


Figure 5. Typical Failure Cases of GPT-4V.

Second, mathematical expressions often have complex structures and hierarchical relationships composed of symbols, constants, variables, etc. For example, $\int_0^\infty x^n e^{-x} dx$ is a typical mathematical expression in advanced mathematics with a complex structure, involving concepts such as integration, exponential functions, and power functions. Understanding such expressions requires strong text parsing abilities.

Third, mathematical expressions are not isolated in problem-solving and theorem proving, but have various logical and numerical relationships. Understanding or generating mathematical expressions with relationships requires logical reasoning and numerical calculation abilities.

8.3 Indirect Proof

Currently, researchers usually employ direct proof approaches when using LLMs in mathematical proofs. Unlike directly proving a proposition P to be *true*, indirect proofs often demonstrate that P cannot be *false*. Indirect proofs are commonly used in mathematics, particularly for proving existence, uniqueness, and propositions that are challenging to prove directly.

Indirect proofs can be further classified into *proof by contradiction* and *proof by contrapositive*. The fundamental principle of *proof by contradiction* is: to prove P to be *true*, first assume P to be *false*, which leads to a contradiction, thereby demonstrating P to be *true*, i.e., $(\neg P \Rightarrow \perp) \Rightarrow P$. One classic example is the proof of *The infinitude of the primes* by the ancient Greek mathematician Euclid using *proof by contradiction*. The basic principle of *proof by contrapositive* is: to prove $P \Rightarrow Q$, one can prove its contrapositive, i.e., $\neg Q \Rightarrow \neg P$. For example, *proof by contrapositive* can easily demonstrate *If n is an integer and n^2 is even, then n is also even*.

Directly applying LLMs for indirect proof may involve modus tollens inference. Empirical studies show that LLMs exhibit the *inverse scaling* phenomenon [2] on such reasoning tasks, which means that though the size of the model and the amount of the training data have increased, the performance of the LLMs decreased. Combining large models with symbolic reasoning also faces challenges such as the need for more skillful strategies in finding reasoning paths. For example, in the indirect proof of *The infinitude of the*

primes, it is necessary to creatively introduce an integer $N = \prod_{i=1}^{n} p_i + 1$, where p_i is the *i*th prime in ascending order, and p_n is the largest prime.

8.4 LLM Benchmark for Solving Mathematical Problems

Although there have been many research works on evaluating the mathematical abilities of LLMs [1, 3-8], constructing a fair, accurate, and comprehensive benchmark for the assessment of mathematical abilities still faces many challenges:

First is the issue of diversity, manifested in the various types, difficulty levels, and solution methods of mathematical problems. To comprehensively assess the mathematical abilities of LLMs, the benchmark should cover a wide range of problem types and difficulties, as well as multiple solution approaches.

Second is the problem of interpretability. Even though LLMs can generate correct answers, if they cannot provide concise and easily understandable solution processes, these answers remain unacceptable. Therefore, there is a need to design indicators and quantitative methods specifically for interpretability.

Third is the issue of data contamination, it means that the evaluation dataset contains data seen during training, leading to evaluation results far higher than the actual performance of the model. For example, evaluation results on 4,550 problems from 30 math and EECS courses required for MIT degree exams indicated that [93] GPT-4 almost passed with full score. But subsequent analysis revealed that significant contamination of the test dataset. Therefore, it is necessary to ensure that the mathematical problems in the evaluation data are not present in the training data of LLMs.

9. CONCLUSION

In this survey, we reviewed methods for solving mathematical problems using LLMs by analyzing relevant papers in the field of LLMs and mathematical problem-solving. Based on our analysis, we propose a two-layer classification system for SMP-LLM. At the first layer, existing researches are classified into four categories based on the approach to problem-solving, including: fine-tuning, prompt engineering, collaboration with symbolic solvers, and collaboration with evaluators/validators. And at the second layer, mathematical problems addressed by existing researches are classified into four categories, including: math word problem, geometry problem, theorem proving, and combinatorial optimization problem.This classification system demonstrates the correlation between solution methods and mathematical problem types of SMP-LLM. We provide a comprehensive analysis of existing works along the dimensions defined by our classification system, and stated the strengths and weaknesses of each approach. Finally, we systematically introduce related datasets in this field and outline future research directions and challenges. We hope our survey will assist other researchers in making further contributions to this area.

AUTHOR CONTRIBUTIONS

Feijuan He: Conceptualization, Methodology, Writing-Original draft preparation (Chapters 1, 2, 8), Writing-Review & Editing. Han Lai: Writing-Original draft preparation (Chapters 3, 4, 9), Writing-Review & Editing. Jiaqi Liu: Writing-Review & Editing (Chapter 5). Bo Wang: Writing-Review & Editing (Chapter 6). Haoran Chen: Writing-Review & Editing (Chapter 6). Haohan Liu: Data curation (Chapter 7), Writing-Review & Editing (Chapter 7). Chenxi Zhang: Data curation (Chapter 7), Writing-Review & Editing (Chapter 7).

ACKNOWLEDGEMENTS

This work was supported by the National Natural Science Foundation of China (No. 62477037), the Shaanxi Provincial Social Science Foundation Project (No. 2024P041), the Youth Innovation Team of Shaanxi Universities "Multi-modal Data Mining and Fusion", Shaanxi Undergraduate and Higher Education Teaching Reform Research Program (No. 23BY195), and Xi'an Jiaotong University City College Research Project (No. 2024Y01).

NON-STANDARD ABBREVIATIONS

SMP-LLM: solving mathematical problems using Large Language Models.

REFERENCES

[1] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt, "Measuring mathematical problem solving with the MATH dataset," in *NeurIPS Datasets and Benchmarks*, 2021.

[2] I. R. McKenzie, A. Lyzhov, M. Pieler, A. Parrish, A. Mueller, A. Prabhu, E. McLean, A. Kirtland, A. Ross, A. Liu, A. Gritsevskiy, D. Wurgaft, D. Kauffman, G. Recchia, J. Liu, J. Cavanagh, M. Weiss, S. Huang, T. F. Droid, T. Tseng, T. Korbak, X. Shen, Y. Zhang, Z. Zhou, N. Kim, S. R. Bowman, and E. Perez, "Inverse scaling: When bigger isn't better," *Trans. Mach. Learn. Res.*, vol. 2023, 2023.

[3] W. Chen, M. Yin, M. Ku, P. Lu, Y. Wan, X. Ma, J. Xu, X. Wang, and T. Xia, "Theoremqa: A theorem-driven question answering dataset," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7889–7901, 2023.

[4] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, pp. 681–694, 2020.

[5] S. Frieder, L. Pinchetti, A. Chevalier, R. Griffiths, T. Salvatori, T. Lukasiewicz, P. Petersen, and J. Berner, "Mathematical capabilities of chatgpt," in *NeurIPS*, 2023.

[6] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark, *et al.*, "Training compute-optimal large language models," *arXiv preprint arXiv:2203.15556*, 2022.

[7] M. Kazemi, H. Alvari, A. Anand, J. Wu, X. Chen, and R. Soricut, "Geomverse: A systematic evaluation of large models for geometric reasoning," *arXiv preprint arXiv:2312.12241*, 2023.

[8] P. Lu, H. Bansal, T. Xia, J. Liu, C. Li, H. Hajishirzi, H. Cheng, K. Chang, M. Galley, and J. Gao, "Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts," in *ICLR*, OpenReview.net, 2024.

[9] F. Xu, Q. Lin, J. Han, T. Zhao, J. Liu, and E. Cambria, "Are large language models really good logical reasoners? a comprehensive evaluation from deductive, inductive and abductive views," *arXiv preprint arXiv:2306.09841*, 2023.

[10] W. Liu, H. Hu, J. Zhou, Y. Ding, J. Li, J. Zeng, M. He, Q. Chen, B. Jiang, A. Zhou, *et al.*, "Mathematical language models: A survey," *arXiv preprint arXiv:2312.07622*, 2023.

[11] D. Gašević, D. Djurić, and V. Devedžić, *Model driven architecture and ontology development*, vol. 10. Springer, 2006.

[12] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[13] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24824–24837, 2022.

[14] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi, *et al.*, "Mathematical discoveries from program search with large language models," *Nature*, pp. 1–3, 2023.

[15] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe, "Let's verify step by step," in *ICLR*, OpenReview.net, 2024.

[16] L. Yu, W. Jiang, H. Shi, J. Yu, Z. Liu, Y. Zhang, J. T. Kwok, Z. Li, A. Weller, and W. Liu, "Metamath: Bootstrap your own mathematical questions for large language models," in *ICLR*, OpenReview.net, 2024.

[17] F. Xu, Z. Wu, Q. Sun, S. Ren, F. Yuan, S. Yuan, Q. Lin, Y. Qiao, and J. Liu, "Symbol-llm: Towards foundational symbol-centric interface for large language models," in *ACL (1)*, pp. 13091–13116, Association for Computational Linguistics, 2024.

[18] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, *et al.*, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[19] W. Ling, D. Yogatama, C. Dyer, and P. Blunsom, "Program induction by rationale generation: Learning to solve and explain algebraic word problems," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 158–167, 2017.

[20] S. An, Z. Ma, Z. Lin, N. Zheng, J.-G. Lou, and W. Chen, "Learning from mistakes makes llm better reasoner," *arXiv preprint arXiv:2310.20689*, 2023.

[21] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[22] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," in *NeurIPS*, 2023.

[23] Z. Yuan, H. Yuan, C. Li, G. Dong, C. Tan, and C. Zhou, "Scaling relationship on learning mathematical reasoning with large language models," *arXiv preprint arXiv:2308.01825*, 2023.

[24] H. Luo, Q. Sun, C. Xu, P. Zhao, J. Lou, C. Tao, X. Geng, Q. Lin, S. Chen, Y. Tang, and D. Zhang, "Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct," in *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24–28, 2025*, OpenReview.net, 2025.

[25] Z. Hu, L. Wang, Y. Lan, W. Xu, E. Lim, L. Bing, X. Xu, S. Poria, and R. K. Lee, "Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models," in *EMNLP*, pp. 5254–5276, Association for Computational Linguistics, 2023.

[26] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[27] J. Gao, R. Pi, J. Zhang, J. Ye, W. Zhong, Y. Wang, L. Hong, J. Han, H. Xu, Z. Li, and L. Kong, "G-llava: Solving geometric problem with multi-modal large language model," in *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24–28, 2025*, OpenReview.net, 2025.

[28] P. Lu, R. Gong, S. Jiang, L. Qiu, S. Huang, X. Liang, and S.-c. Zhu, "Inter-gps: Interpretable geometry problem solving with formal language and symbolic reasoning," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 6774–6786, 2021.

[29] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," *Advances in neural information processing systems*, vol. 36, pp. 34892–34916, 2023.

[30] Z. Liang, T. Yang, J. Zhang, and X. Zhang, "Unimath: A foundational and multimodal mathematical reasoner," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7126–7133, 2023.

[31] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma, A. Webson, S. S. Gu, Z. Dai, M. Suzgun, X. Chen, A. Chowdhery, A. Castro-Ros, M. Pellat, K. Robinson, D. Valter, S. Narang, G. Mishra, A. Yu, V. Y. Zhao, Y. Huang, A. M. Dai, H. Yu, S. Petrov, E. H. Chi, J. Dean, J. Devlin, A. Roberts, D. Zhou, Q. V. Le, and J. Wei, "Scaling instruction-finetuned language models," *J. Mach. Learn. Res.*, vol. 25, pp. 70:1–70:53, 2024.

[32] A. Van Den Oord, O. Vinyals, *et al.*, "Neural discrete representation learning," *Advances in neural information processing systems*, vol. 30, 2017.

[33] A. Razavi, A. van den Oord, and O. Vinyals, "Generating diverse high-fidelity images with vq-vae-2," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 14866–14876, 2019.

[34] A. Patel, S. Bhattamishra, and N. Goyal, "Are nlp models really able to solve simple math word problems?," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2080–2094, 2021.

[35] J. Chen, J. Tang, J. Qin, X. Liang, L. Liu, E. Xing, and L. Lin, "Geoqa: A geometric question answering benchmark towards multimodal numerical reasoning," in *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 513–523, 2021.

[36] P. Lu, L. Qiu, K.-W. Chang, Y. N. Wu, S.-C. Zhu, T. Rajpurohit, P. Clark, and A. Kalyan, "Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning," in *The Eleventh International Conference on Learning Representations*, 2022.

[37] A. Amini, S. Gabriel, S. Lin, R. Koncel-Kedziorski, Y. Choi, and H. Hajishirzi, "Mathqa: Towards interpretable math word problem solving with operation-based formalisms," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2357–2367, 2019.

[38] J. Chen, T. Li, J. Qin, P. Lu, L. Lin, C. Chen, and X. Liang, "Unigeo: Unifying geometry logical reasoning via reformulating mathematical expression," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 3313–3323, 2022.

[39] W. X. Zhao, K. Zhou, Z. Gong, B. Zhang, Y. Zhou, J. Sha, Z. Chen, S. Wang, C. Liu, and J.-R. Wen, "Jiuzhang: A chinese pre-trained language model for mathematical problem understanding," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4571–4581, 2022.

[40] X. Zhao, K. Zhou, B. Zhang, Z. Gong, Z. Chen, Y. Zhou, J.-R. Wen, J. Sha, S. Wang, C. Liu, *et al.*, "Jiuzhang 2.0: A unified chinese pre-trained language model for multi-task mathematical problem solving," in

Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pp. 5660–5672, 2023.

[41] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, pp. 24824–24837, 2022.

[42] Z. Zhang, A. Zhang, M. Li, and A. Smola, "Automatic chain of thought prompting in large language models," in *The Eleventh International Conference on Learning Representations*, 2022.

[43] X. Huang, L. L. Zhang, K.-T. Cheng, and M. Yang, "Boosting llm reasoning: Push the limits of few-shot learning with reinforced in-context pruning," *arXiv preprint arXiv:2312.08901*, 2023.

[44] Y. Fu, H. Peng, A. Sabharwal, P. Clark, and T. Khot, "Complexity-based prompting for multi-step reasoning," in *ICLR*, OpenReview.net, 2023.

[45] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[46] S. Imani, L. Du, and H. Shrivastava, "Mathprompter: Mathematical reasoning using large language models," in *ACL (industry)*, pp. 37–42, Association for Computational Linguistics, 2023.

[47] X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," in *The Eleventh International Conference on Learning Representations*, 2022.

[48] D. Zhou, N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. V. Le, *et al.*, "Least-to-most prompting enables complex reasoning in large language models," in *The Eleventh International Conference on Learning Representations*, 2022.

[49] H. Kautz, "The third ai summer: Aaai robert s. engelmore memorial lecture," *AI Magazine*, vol. 43, no. 1, pp. 105–125, 2022.

[50] W. Chen, X. Ma, X. Wang, and W. W. Cohen, "Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks," *Trans. Mach. Learn. Res.*, vol. 2023, 2023.

[51] O. Wojciech Zaremba, Greg Brockman, "Openai codex," 2021.

[52] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig, "Pal: Program-aided language models," in *International Conference on Machine Learning*, pp. 10764–10799, PMLR, 2023.

[53] R. Yamauchi, S. Sonoda, A. Sannai, and W. Kumagai, "Lpml: Llm-prompting markup language for mathematical reasoning," *arXiv preprint arXiv:2309.13078*, 2023.

[54] J. He-Yueya, G. Poesia, R. E. Wang, and N. D. Goodman, "Solving math word problems by combining language models with symbolic solvers," *arXiv preprint arXiv:2304.09102*, 2023.

[55] W. Wu, L. Zhang, J. Liu, X. Tang, Y. Wang, S. Wang, and Q. Wang, "E-gps: Explainable geometry problem solving via top-down solver and bottom-up generator," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13828–13837, 2024.

[56] T. H. Trinh, Y. Wu, Q. V. Le, H. He, and T. Luong, "Solving olympiad geometry without human demonstrations," *Nature*, vol. 625, no. 7995, pp. 476–482, 2024.

[57] S.-C. Chou, X.-S. Gao, and J.-Z. Zhang, "A deductive database approach to automated geometry theorem proving and discovering," *Journal of Automated Reasoning*, vol. 25, no. 3, pp. 219–246, 2000.

[58] G. Irving, C. Szegedy, A. A. Alemi, N. Eén, F. Chollet, and J. Urban, "Deepmath-deep sequence models for premise selection," *Advances in neural information processing systems*, vol. 29, 2016.

[59] K. Yang, A. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. J. Prenger, and A. Anandkumar, "Leandojo: Theorem proving with retrieval-augmented language models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[60] T. mathlib Community, "The lean mathematical library," in *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, p. 367–381, Association for Computing Machinery, 2020.

[61] A. Thakur, Y. Wen, and S. Chaudhuri, "A language-agent approach to formal theorem-proving," *arXiv preprint arXiv:2310.04353*, 2023.

[62] G. Huet, G. Kahn, and C. Paulin-Mohring, "The coq proof assistant a tutorial," *Rapport Technique*, vol. 178, 1997.

[63] L. de Moura, S. Kong, J. Avigad, F. Van Doorn, and J. von Raumer, "The lean theorem prover (system description)," in *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1–7, 2015, Proceedings 25*, pp. 378–388, Springer, 2015.

[64] Y. Wu, A. Q. Jiang, W. Li, M. Rabe, C. Staats, M. Jamnik, and C. Szegedy, "Autoformalization with large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 32353–32368, 2022.

[65] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, *et al.*, "Palm: Scaling language modeling with pathways," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023.

[66] T. Nipkow, M. Wenzel, and L. C. Paulson, *Isabelle/HOL: a proof assistant for higher-order logic*. Springer, 2002.

[67] S. Welleck and R. Saha, "Llmstep: Llm proofstep suggestions in lean," in *The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23*, 2023.

[68] S. Biderman, H. Schoelkopf, Q. G. Anthony, H. Bradley, K. O'Brien, E. Hallahan, M. A. Khan, S. Purohit, U. S. Prashanth, E. Raff, *et al.*, "Pythia: A suite for analyzing large language models across training and scaling," in *International Conference on Machine Learning*, pp. 2397–2430, PMLR, 2023.

[69] Y. Li, Z. Lin, S. Zhang, Q. Fu, B. Chen, J.-G. Lou, and W. Chen, "Making language models better reasoners with step-aware verifier," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5315–5333, 2023.

[70] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," in *International Conference on Learning Representations*, 2020.

[71] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, S. Gupta, B. P. Majumder, K. Hermann, S. Welleck, A. Yazdanbakhsh, and P. Clark, "Self-refine: Iterative refinement with self-feedback," in *NeurIPS*, 2023.

[72] A. Zhou, K. Wang, Z. Lu, W. Shi, S. Luo, Z. Qin, S. Lu, A. Jia, L. Song, M. Zhan, and H. Li, "Solving challenging math word problems using GPT-4 code interpreter with code-based self-verification," in *ICLR*, OpenReview.net, 2024.

[73] S. Polu and I. Sutskever, "Generative language modeling for automated theorem proving," *arXiv preprint arXiv:2009.03393*, 2020.

[74] E. First, M. Rabe, T. Ringer, and Y. Brun, "Baldur: Whole-proof generation and repair with large language models," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1229–1241, 2023.

[75] F. Liu, X. Tong, M. Yuan, and Q. Zhang, "Algorithm evolution using large language model," *CoRR*, vol. abs/2311.15249, 2023.

[76] F. Liu, X. Tong, M. Yuan, X. Lin, F. Luo, Z. Wang, Z. Lu, and Q. Zhang, "An example of evolutionary computation+ large language model beating human: Design of efficient guided local search," *arXiv preprint arXiv:2401.02051*, 2024.

[77] S. Liu, C. Chen, X. Qu, K. Tang, and Y. Ong, "Large language models as evolutionary optimizers," in *CEC*, pp. 1–8, IEEE, 2024.

[78] DeepSeek-AI, D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, X. Zhang, X. Yu, Y. Wu, Z. F. Wu, Z. Gou, Z. Shao, Z. Li, Z. Gao, A. Liu, B. Xue, B. Wang, B. Wu, B. Feng, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Ding, H. Xin, H. Gao, H. Qu, H. Li, J. Guo, J. Li, J. Wang, J. Chen, J. Yuan, J. Qiu, J. Li, J. L. Cai, J. Ni, J. Liang, J. Chen, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Zhao, L. Wang, L. Zhang, L. Xu, L. Xia, M. Zhang, M. Zhang, M. Tang, M. Li, M. Wang, M. Li, N. Tian, P. Huang, P. Zhang, Q. Wang, Q. Chen, Q. Du, R. Ge, R. Zhang, R. Pan, R. Wang, R. J. Chen, R. L. Jin, R. Chen, S. Lu, S. Zhou, S. Chen, S. Ye, S. Wang, S. Yu, S. Zhou, S. Pan, and S. S. Li, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *CoRR*, vol. abs/2501.12948, 2025.

[79] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. Li, Y. Wu, *et al.*, "Deepseekmath: Pushing the limits of mathematical reasoning in open language models," *arXiv preprint arXiv:2402.03300*, 2024.

[80] Z. Xi, W. Chen, B. Hong, S. Jin, R. Zheng, W. He, Y. Ding, S. Liu, X. Guo, J. Wang, *et al.*, "Training large language models for reasoning through reverse curriculum reinforcement learning," in *International Conference on Machine Learning*, pp. 54030–54048, PMLR, 2024.

[81] S.-Y. Miao, C.-C. Liang, and K.-Y. Su, "A diverse corpus for evaluating and developing english math word problem solvers," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 975–984, 2020.

[82] J. Cao and J. Xiao, "An augmented benchmark dataset for geometric question answering through dual parallel text encoding," in *Proceedings of the 29th International Conference on Computational Linguistics*, pp. 1511–1520, 2022.

[83] M. Zhang, F. Yin, and C. Liu, "A multi-modal neural geometric solver with textual clauses parsed from diagram," in *IJCAI*, pp. 3374–3382, ijcai.org, 2023.

[84] J. Zhang, Z. Li, M. Zhang, F. Yin, C. Liu, and Y. Moshfeghi, "Geoeval: Benchmark for evaluating llms and multi-modal models on geometry problem-solving," in *ACL (Findings)*, pp. 1258–1276, Association for Computational Linguistics, 2024.

[85] F. Wiedijk, "Formalizing 100 theorems," 2023.

[86] K. Zheng, J. M. Han, and S. Polu, "minif2f: a cross-system benchmark for formal olympiad-level mathematics," in *International Conference on Learning Representations*, 2021.

[87] N. Megill and D. A. Wheeler, *Metamath: a computer language for mathematical proofs*. Lulu. com, 2019.

[88] C. Liu, J. Shen, H. Xin, Z. Liu, Y. Yuan, H. Wang, W. Ju, C. Zheng, Y. Yin, L. Li, *et al.*, "Fimo: A challenge formal dataset for automated theorem proving," *arXiv preprint arXiv:2309.04295*, 2023.

[89] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[90] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European conference on computer vision*, pp. 213–229, Springer, 2020.

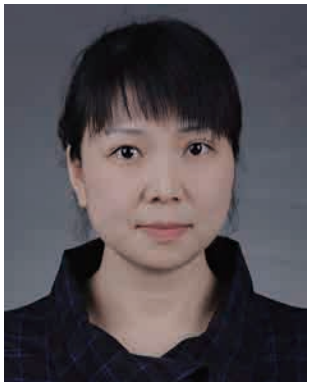
[91] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "Centernet: Keypoint triplets for object detection," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6569–6578, 2019.

[92] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *13th European Conference Computer Vision*, pp. 740–755, Springer, 2014.

[93] S. J. Zhang, S. Florin, A. N. Lee, E. Niknafs, A. Marginean, A. Wang, K. Tyser, Z. Chin, Y. Hicke, N. Singh, *et al.*, "Exploring the mit mathematics and eecs curriculum using large language models," *arXiv preprint arXiv:2306.08997*, 2023.

[94] H. Wang, H. Xin, C. Zheng, Z. Liu, Q. Cao, Y. Huang, J. Xiong, H. Shi, E. Xie, J. Yin, Z. Li, and X. Liang, "Lego-prover: Neural theorem proving with growing libraries," in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7–11, 2024*, OpenReview.net, 2024.

AUTHOR BIOGRAPHY



Feijuan He is an Associate Professor at Xi'an Jiaotong University City College, holding a master's degree from The Hong Kong Polytechnic University. Her research interests include educational data mining, natural language processing, and large language models.

Han Lai is a graduate student in the School of Computer Science and Technology at Xi'an Jiaotong University. Her research focuses on enhancing the reasoning abilities of Large Language Models.

Liu Jiaqi is a postgraduate student specializing in pediatric traditional Chinese medicine at Dongfang Hospital of Beijing University of Chinese Medicine. Her research focuses on medical data mining.

Bo Wang is a Lecturer at Xi'an Jiaotong University City College, holding a master's degree from Chongqing University. His research interests include industrial big data mining and analysis, and pattern recognition in the healthcare field.

Haoran Chen is an undergraduate student majoring in Computer Science at Xi'an Jiaotong University City College. His research interests include data mining, computer vision, and machine learning.

Haohan Liu is an undergraduate student majoring in Computer Science at Xi'an Jiaotong University City College. His research interests focus on learning data analysis and machine learning.

Chenxi Zhang is an undergraduate student majoring in Computer Science at Xi'an Jiaotong University City College. Her research interests include data mining and machine learning.

APPENDIX A. DATA ANALYSIS

If not specially specified, the evaluation metric for each dataset is accuracy, the unit is %.

Table A.10: Math Word Problems.

Model	GSK8K	MATH	SVAMP
Fine-tuning			
MetaMath-7B/13B/70B [16]	66.5/72.3/82.3	19.8/22.4/26.6	-
Symbol-LLMbase-7B/13B [17]	61.14/68.69	28.24/33.39	-
Symbol-LLMinstruct-7B/13B	59.36/65.58	26.54/31.32	-
LLaMA-2-7B/13B/70B + Learning From Mistakes [20]	54.1/64.2/83.5	9.4/12.6/25.0	-/-/81.6
LLaMA-65B + Learning From Mistakes	77.9	20.8	72.8
MetaMath-13B/70B + Learning From Mistakes	73.2/85.4	22.7/26.9	-
WizardMath-7B/13B/70B + Learning From Mistakes	55.9/73.2/84.2	11.9/22.7/27.1	-
LLaMa2-13B + RFT-U13B [23]	55.4	-	-
LLaMa-7B/13B + LorA [25]	37.5/47.5	-	52.1/54.6
Prompt Engineering			
Auto-CoT [42]	47.9	-	69.5
Code-davinci-002 + Complex CoT [44]	66.6	-	-
Code-davinci-002 + Vote Complex	82.9	-	-
LLaMA2-7B/13B/70B + CoT-Max [43]	15.92/32.37/59.59	-	-
Code-davinci-002 + Self-consistency [47]	78.0	-	86.8
Code-davinci-002 + LeasLLaMA2-7B/13B/70Bt-to-Most [48]	62.39	-	-
Collaborating with Symbolic Solvers			
PoT + GPT4 [50]	97.2	-	97.4
Codex + PAL [52]	72.0	-	79.4
DECLARATIVE3-shot + principles + SymPy [54]	69.4 ± 0.65	-	-
LPML + GPT-3.5 [53]	76.6	60.0	-

Table A.11: Math Word Problems.

Model	GSK8K	MATH	SVAMP
Collaborating with Evaluators/Validators			
Code-davinci-002 + DIVERSE [69]	82.3	-	87.0
GPT-4 + SELF-REFINE [71]	93.1	-	-
GPT-4-Code + CSV [72]	-	73.54	-
GPT-4-Code + CSV + Voting	-	84.32	-
WizardMath-GPT2-Small/Medium/Large/XL [24]	26.4/38.7/50.1/58.9	12.3/15.6/21.2/25.4	-
WizardMath-Qwen2.5-7B	94.0	74.5	-
WizardMath-Qwen2.5-Math-7B	93.9	77.8	-
WizardMath-DeepSeekMath-7B	91.0	64.6	-
WizardMath-Llama-3-1B/3B/8B	63.3/85.5/90.3	33.5/49.9/58.8	-
WizardMath-Llama-2-7B/13B/70B	84.1/89.7/92.8	43.5/50.6/58.6	-
DeepSeekMath Corpus [79]	23.8	13.6	-
DeepSeekMath-Base-7B	64.2	36.2	-
DeepSeekMath-Instruct-7B (Chain-of-Thought Reasoning)	82.9	46.8	-
DeepSeekMath-RL-7B (Chain-of-Thought Reasoning)	88.2	51.7	-
DeepSeekMath-Instruct-7B (Tool-Integrated Reasoning)	83.7	57.4	-
DeepSeekMath-RL-7B (Tool-Integrated Reasoning)	86.7	58.8	-
Llama2-Base-7B + R^3 [80]	50.49	-	64.40

Table A.12: Geometry Problems.

Model	SVAMP	GeoQA	MathVista	Geometry3K
Fine-tuning				
UniMath-T5-base 37.3 [30]	49.6	-	-	
UniMath-Flan-T5-base	41.8	50.0	-	-
G-LLaVA-7B/13B [27]	-	64.2/67.0	53.4/56.7	-
Collaborating with Symbolic Solvers				
Inter-GPS [28]	-	-	-	57.5

Table A.14: Theorem Proving.

Model	miniF2F-valid	miniF2F-test	LeanDojo Benchmark
Prompt Engineering			
Thor after 2 expert iterations (M2) [64]	37.3	35.2	-
LEGO-Prover + ChatGPT [94]	57.0	50.0	-
Collaborating with Symbolic Solvers			
COPRA + GPT-3.5 [61]	-	22.13	-
COPRA + GPT-4	-	23.36	-
ReProver [59]	-	26.5	51.2/26.3
LLMSTEP (Pythia-2.8b) [67]	26.2	27.9	-

Table A.15: Combinatorial Optimization Problems.

Model	TSP20 Gap	TSP20 Gap	TSP100 Gap	TSP200 Gap	TSP500 Gap	TSP1000 Gap
Collaborating with Evaluators/Validators						
AEL + GPT-3.5-turbo [75]	11.2	16.8	20.0	21.8	23.1	22.8
AEL + GPT-4	6.2	11.1	10.5	11.2	12.8	12.8
AEL-GLS [76]	0.000	0.000	0.032	-	-	-

Table A.16: Combinatorial Optimization Problems.

Model	OR1	OR2	OR3	OR4	Weibull 5k	Weibull 10k	Weibull 100k
Collaborating with Evaluators/Validators							
FunSearch [14]	5.30	4.19	3.11	2.47	0.68	0.32	0.03

Table A.17: Combinatorial Optimization Problems.

Model	rue-10 Gap	rue-15 Gap	rue-20 Gap	rue-25 Gap
Collaborating with Evaluators/Validators				
LMEA [77]	0.00% ± 0.00%	0.06% ± 0.06%	3.94% ± 1.54%	18.72% ± 3.31%

Table A.18: Combinatorial Optimization Problems.

Model	clu-10 Gap	clu-15 Gap	clu-20 Gap	clu-25 Gap
Collaborating with Evaluators/Validators				
LMEA	0.00% ± 0.00%	0.11% ± 0.11%	4.05% ± 0.69%	10.06% ± 1.69%