Training and Evaluating Language Models with Template-based Data Generation

Yifan Zhang[1] **et al.**
[1]IIIS, Tsinghua University
zhangyif21@mails.tsinghua.edu.cn

February 2nd, 2024

Abstract

The rapid advancement of large language models (LLMs) such as GPT-3, PaLM, and Llama has significantly transformed natural language processing, showcasing remarkable capabilities in understanding and generating language. However, these models often struggle with tasks requiring complex reasoning, particularly in mathematical problem-solving, due in part to the scarcity of large-scale, high-quality, domain-specific datasets necessary for training sophisticated reasoning abilities. To address this limitation, we introduce *Template-based Data Generation* (TDG), a novel approach that leverages LLMs (GPT-4) to automatically generate parameterized meta-templates, which are then used to synthesize a vast array of high-quality problems and solutions. Leveraging TDG, we create **TemplateMath Part I: TemplateGSM**, a dataset comprising over 7 million synthetically generated grade school math problems—each accompanied by code-based and natural language solutions—with the potential to generate an effectively unlimited number more. This dataset alleviates the scarcity of large-scale mathematical datasets and serves as a valuable resource for pre-training, fine-tuning, and evaluating LLMs in mathematical reasoning. Our method not only enables the generation of virtually infinite data but also elevates data augmentation to a new level by using GPT-4 for meta-template generation, ensuring diverse and high-quality problem structures. The TemplateMath Part I: TemplateGSM dataset is publicly available at `https://huggingface.co/datasets/math-ai/TemplateGSM`.[1]

1 Introduction

Large language models (LLMs) have revolutionized natural language processing (NLP), exhibiting unprecedented capabilities in language understanding and generation. Models such as GPT-3 (Brown et al., 2020), PaLM (Chowdhery et al., 2022), and Llama (Touvron et al., 2023) have achieved remarkable success across various NLP tasks, including machine translation, summarization, and question answering.

Despite these advancements, LLMs often struggle with tasks requiring complex reasoning and precise problem-solving skills, particularly in mathematical domains (Hendrycks et al., 2021; Cobbe et al., 2021). Mathematical reasoning poses unique challenges due to its reliance on rigorous logic, structured methodologies, and the necessity for exact solutions. Existing mathematical datasets are limited in both size and diversity, hindering models' ability to generalize across a wide range of problems (Paster et al., 2023; Azerbayev et al., 2023). The scarcity of large-scale, high-quality mathematical datasets is a significant barrier to developing LLMs capable of sophisticated mathematical reasoning.

[1]The code is available at `https://github.com/iiis-ai/TemplateMath`.

To address this challenge, we introduce *Template-based Data Generation* (TDG), a method that systematically generates an extensive variety of mathematical problems and corresponding solutions by leveraging parameterized templates. Crucially, we elevate data augmentation to a new level by employing GPT-4 to automatically generate these meta-templates, which serve as foundational structures for problem generation. By utilizing GPT-4's advanced language understanding and generation capabilities, we create a diverse set of templates that capture a wide range of mathematical problem types.

Using TDG, we present **TemplateGSM**, a dataset of over 7 million synthetically generated grade school math problems, each paired with verified solutions in both code-based and natural language formats. By automating the generation and verification of problems and solutions, our approach ensures high-quality data and precise supervision, facilitating the development of LLMs with advanced mathematical reasoning capabilities.

Our main contributions are:

- **Template-based Data Generation (TDG)**: We introduce TDG, a scalable method for generating an effectively infinite amount of high-quality, domain-specific data using parameterized templates generated by GPT-4.
- **Elevated Data Augmentation**: By leveraging GPT-4 to create meta-templates, we advance data augmentation to a new level, ensuring a diverse and rich set of problem structures for data synthesis.
- **Creation of TemplateGSM Dataset**: We develop TemplateGSM, a dataset comprising over 7 million synthetically generated math problems with verified solutions, addressing the scarcity of large-scale mathematical datasets.
- **Enhancement of LLM Performance**: We demonstrate the efficacy of the generated data for pre-training, fine-tuning, and evaluating LLMs, enhancing their performance in mathematical reasoning tasks.
- **Precise Supervision through Code Execution**: We provide insights into how TDG offers precise supervision through code execution and verification, promoting the development of models with improved understanding and problem-solving abilities.

2 Template-based Data Generation

2.1 Overview

Template-based Data Generation (TDG) is a method designed to systematically produce a vast array of mathematical problems along with their corresponding solutions by leveraging parameterized templates. To elevate data augmentation to a new level, we employ GPT-4 to generate these meta-templates, capturing a wide variety of problem structures and linguistic styles. By varying parameters within these GPT-4-generated templates, TDG ensures both scalability and quality in the generated data. This approach enables the creation of diverse and complex problem sets, which are essential for training and evaluating large language models in mathematical reasoning tasks.

2.2 Methodology

The TDG process involves several key components that work together to generate high-quality mathematical datasets:

2.2.1 Generation of Meta-Templates with GPT-4

We begin by utilizing GPT-4 to generate meta-templates that capture the underlying structure of various types of mathematical problems. GPT-4's advanced language generation capabilities allow us to produce a diverse set of templates that encompass a wide range of mathematical concepts and problem types. These templates include placeholders for variable components such as names, items, quantities, dates, and locations. By harnessing GPT-4, we ensure that the templates are linguistically diverse and contextually rich, which contributes to the overall quality and diversity of the dataset.

2.2.2 Parameter Generation

To instantiate the templates, we develop functions that generate parameters satisfying specific conditions, ensuring the solvability and validity of the problems. This step is crucial for maintaining the integrity of the dataset, as it guarantees that each generated problem is mathematically sound and that its solution can be accurately computed. Parameters are carefully selected to avoid trivial or overly complex problems, striking a balance that is appropriate for the target educational level.

2.2.3 Problem Instantiation

The generated parameters are then substituted into the GPT-4-generated meta-templates to create specific problem statements. Each instantiated problem is unique in its details but retains the structural characteristics defined by the template. This process allows us to produce a wide variety of problems that are diverse in content yet consistent in form, which is beneficial for training models to recognize and solve different problem types.

2.2.4 Solution Generation and Verification

For each problem, we generate solution code—typically in Python—that can automatically solve the problem. This code includes detailed comments and variable names that reflect the specifics of the problem, enhancing readability and transparency. By executing the solution code, we obtain the results and verify the correctness of the solutions. This automated verification is essential for ensuring the accuracy and reliability of the dataset.

In addition to the code-based solutions, we also generate natural language explanations that describe the solution steps without using code. These explanations translate the computational procedures into plain language, making the solutions accessible and understandable to individuals without programming expertise. This dual representation enriches the dataset and provides multiple modalities for training language models.

2.3 Illustrative Example

An illustrative example of our TDG method is presented in Figure 1. The code snippet demonstrates how we generate problems involving sales over two consecutive months. The meta-template for this problem type was generated using GPT-4, capturing a realistic scenario that can be varied through parameter substitution. We include lists of random terms such as names, items, months, and locations to create diverse and realistic problem contexts. This randomness introduces variability and prevents the dataset from becoming repetitive, which helps in training models to generalize better.

2.4 Generated Problem and Solution Example

To illustrate the output of our TDG method, consider the following example generated using a GPT-4-produced template and the code from Figure 1:

Problem:

Alex Johnson sold 120 books in March, 2021 at The Bookstore in Orange County, California. In April, they sold 80% of the amount sold in the previous month. How many books did Alex Johnson sell in total during March and April?

Solution:

To solve this problem, we first note that Alex sold 120 books in March. In April, they sold 80% of the March sales, which amounts to $0.8 \times 120 = 96$ books. Therefore, the total number of books sold during March and April is $120 + 96 = 216$ books.

Code-based Solution:

```
# Number of books sold by Alex Johnson in March, 2021
books_sold_in_March = 120

# Sales ratio for the next month
books_ratio = 0.8

# Calculating the amount of books sold in April
subsequent_books_sold_in_March = books_sold_in_March * books_ratio
```

```
def generate_problem_and_solution_code():
    # Lists of random terms
    months = ["January and February", ..., "December and January"]

    # Get initial amount and subsequent ratio
    initial_amount, subsequent_ratio = get_params_combination()

    # Randomly select terms
    name = random.choice(first_names) + ' ' + random.choice(last_names)
    item = random.choice(items)
    month = random.choice(months)
    year = random.randint(2003, 2023)
    place = random.choice(places)
    county = random.choice(us_counties)
    county = county['CountyName'] + ", " + county["StateName"]

    # Construct problem statement
    problem_statement = f"{name} sold {initial_amount} {item} in {month.split(' and ')
     [0]}, {year} at {place} in {county}. "
    problem_statement += f"In {month.split(' and ')[1]}, they sold {subsequent_ratio
     *100:.0f}% of the amount sold in the previous month. "
    problem_statement += f"How many {item} did {name} sell in total during {month}?"

    # Generate solution code
    sales_var = f"{item.replace(' ', '_')}_sold_in_{month.split(' ')[0]}"
    ratio_var = f"{item.replace(' ', '_')}_ratio"
    total_var = f"total_{item.replace(' ', '_')}"

    solution_code = f"""# Number of {item} sold by {name} in {month.split(' and ')[0]}, {
     year}
{sales_var} = {initial_amount}

# Sales ratio for the next month
{ratio_var} = {subsequent_ratio}

# Calculating the amount of {item} sold in {month.split(' and ')[1]}
subsequent_{sales_var} = {sales_var} * {ratio_var}

# Calculating the total number of {item} sold during {month}
{total_var} = {sales_var} + subsequent_{sales_var}

result = {total_var}
"""

    # Execute the solution code
    exec_globals = {}
    exec(solution_code, {}, exec_globals)
    result = round(exec_globals['result'])

    # Generate the solution without code
    solution_wocode = f"{name} sold {initial_amount} {item} in {month.split(' and ')[0]},
     {year}. "
    solution_wocode += f"In {month.split(' and ')[1]}, they sold {subsequent_ratio*100:.0f
     }% of the amount sold in the previous month, which is {round(subsequent_ratio*
     initial_amount)} {item}. "
    solution_wocode += f"In total, {name} sold {initial_amount} + {int(subsequent_ratio*
     initial_amount)} = {result} {item} during {month}."

    return problem_statement, solution_code, result, solution_wocode
```

Figure 1: An illustrative example of our Template-based Data Generation (TDG) method. The code demonstrates how variable parameters are used to generate unique problem statements and corresponding solutions based on GPT-4-generated meta-templates.

```
# Calculating the total number of books sold during March and April
total_books = books_sold_in_March + subsequent_books_sold_in_March

result = total_books
```

This code provides a computational solution to the problem, ensuring accuracy and allowing for automatic verification of the result.

2.5 Advantages of TDG

The TDG method offers several significant advantages that make it particularly effective for generating large-scale mathematical datasets:

- **Scalability**: TDG enables the generation of an effectively infinite amount of data by varying parameters within GPT-4-generated templates. This scalability is crucial for training large language models that require vast amounts of data.
- **Quality Assurance**: By using code execution for solution verification, we ensure that each problem has a correct and reliable solution. This precise supervision enhances the quality of the dataset and the performance of models trained on it.
- **Diversity**: The use of GPT-4 to generate meta-templates introduces a wide variety of problem structures and linguistic styles, enhancing the diversity of the dataset. This diversity helps models to generalize better to new and unseen problems.
- **Elevated Data Augmentation**: By incorporating GPT-4 into the template generation process, we elevate data augmentation to a new level, enabling the synthesis of data that is both varied and high-quality.

3 TemplateMath Part I: TemplateGSM Dataset

3.1 Dataset Construction

Building upon the TDG method, we have developed **TemplateGSM**, a dataset consisting of over 7 million grade school math problems. Each problem is paired with both a code-based solution and a natural language explanation. The problems cover a wide range of mathematical topics suitable for grade school levels, including arithmetic operations, fractions, percentages, and basic algebra. The meta-templates used to generate these problems were created using GPT-4, ensuring a rich diversity in problem structures and linguistic expressions. This comprehensive coverage ensures that the dataset can be used to train models on various problem types and difficulty levels.

3.2 Dataset Statistics

The key statistics of the TemplateGSM dataset are presented in Table 1. With 7,473,000 problems generated from 7,473 unique GPT-4-generated templates, the dataset offers extensive diversity in both problem structures and content.

Table 1: Statistics of the TemplateGSM Dataset

Metric	Value
Number of source templates	7,473
Total number of problems	7,473,000
Problem length range (tokens)	[85, 1151]
Code solution length range (tokens)	[102, 1835]
Natural language solution length range (tokens)	[36, 1392]

The average lengths of problems and solutions indicate that the dataset provides substantial context and detailed explanations, which are beneficial for training language models to understand and solve complex reasoning tasks.

3.3 Dataset Availability

TemplateGSM is publicly available and can be accessed at `https://huggingface.co/datasets/math-ai/TemplateGSM`. The code used for data generation is also provided at `https://github.com/iiis-ai/TemplateMath`. By sharing both the dataset and the generation code, we enable researchers and practitioners to reproduce our results, extend the dataset, and apply the TDG method to other domains or problem types.

3.4 Applications and Impact

The TemplateGSM dataset serves as a valuable resource for pre-training, fine-tuning, and evaluating large language models in mathematical reasoning tasks. By addressing the data scarcity problem, it

facilitates the development of models capable of sophisticated reasoning. The inclusion of GPT-4-generated templates introduces a level of diversity and naturalness in problem statements that closely mimic human-crafted problems. We anticipate that TemplateGSM and TemplateMath will contribute to advancements in AI research focused on reasoning and problem-solving.

4 Related Work

Mathematical Datasets. The development of mathematical datasets has played a crucial role in advancing AI research, particularly in mathematical reasoning and problem-solving. Early datasets like AQUA-RAT (Ling et al., 2017) provided annotated question-answer pairs for arithmetic word problems. The MATH dataset (Hendrycks et al., 2021) comprises over 12,500 challenging competition-level problems, serving as a benchmark for evaluating mathematical reasoning abilities. However, its limited size restricts its utility for training large models.

To expand available resources, Paster et al. (2023) introduced OpenWebMath, filtering web data to collect mathematical content. While it offers a larger dataset, quality control remains challenging due to the noisy nature of web data. Proof-Pile (Azerbayev et al., 2023) aggregates informal mathematical texts but lacks structured problem-solution pairs necessary for supervised learning.

Our work builds upon these efforts by providing a significantly larger and more diverse dataset of mathematical problems with verified solutions, addressing the need for high-quality training data in mathematical reasoning.

Training LLMs on Mathematical Tasks. Base LLMs trained on vast corpora have demonstrated impressive language capabilities (Brown et al., 2020; Touvron et al., 2023). However, their performance on mathematical tasks is limited due to the scarcity of domain-specific training data. Recent studies have explored fine-tuning LLMs on mathematical datasets through continual pre-training (Lewkowycz et al., 2022; Azerbayev et al., 2023) or supervised fine-tuning (SFT) (Yu et al., 2023; Yue et al., 2023; Weng et al., 2023).

Continual pre-training involves further training on mathematical texts, enhancing models' familiarity with mathematical language but not necessarily improving problem-solving skills. SFT approaches fine-tuning models on curated question-answer pairs but is constrained by the availability of high-quality datasets.

Our TDG method enables the generation of extensive, high-quality problem-solution pairs, providing a valuable resource for both pre-training and fine-tuning LLMs, potentially leading to significant improvements in mathematical reasoning performance.

Data Generation Techniques. Data augmentation and synthetic data generation have been widely used to improve model performance in various domains (Feng et al., 2021). In mathematical problem-solving, methods like problem recombination (Koncel-Kedziorski et al., 2015), and question rephrasing (Yu et al., 2023) have been explored but on a much smaller scale.

Our TDG approach differs by offering a systematic and scalable method to generate an effectively infinite number of high-quality problems, coupled with more precise solution verification through code execution.

5 Conclusion

We have introduced *Template-based Data Generation* (TDG), a novel approach for generating large-scale, high-quality mathematical datasets through parameterized templates generated by GPT-4. Utilizing TDG, we created **TemplateGSM**, a dataset of over 7 million synthetically generated grade school math problems with verified solutions in both code and natural language formats.

Our extensive experiments demonstrate that TemplateGSM significantly enhances the mathematical reasoning capabilities of LLMs when used for pre-training and fine-tuning. The precise supervision offered by code execution and verification ensures the reliability of the data, fostering the development of models with improved understanding and problem-solving abilities.

By leveraging GPT-4 to generate meta-templates, we have elevated data augmentation to a new level, introducing greater diversity and richness in the generated data. We believe that TDG and the TemplateGSM dataset will contribute substantially to advancing research in mathematical reasoning with LLMs. By addressing the data scarcity problem, our work opens new avenues for developing models capable of complex reasoning tasks.

Limitations. While TDG and TemplateGSM offer substantial benefits, there are limitations. One limitation is **template bias**, where models may become biased toward the structures present in the GPT-4-generated templates. Additionally, the generated problems are primarily at the grade school level, so extending TDG to higher-level mathematics requires careful template design, reflecting challenges in addressing **complexity levels**. Moreover, although GPT-4 introduces linguistic diversity, the style may still differ from human-authored educational materials, indicating a limitation in **authenticity**.

Future Work. Future research directions include expanding template coverage by developing templates for more advanced mathematical topics, potentially leveraging even more sophisticated LLMs or collaborative human-AI template generation. Integrating augmentation techniques, such as using language models to rephrase and compose problems, can increase linguistic diversity further. Extending TDG to generate problems in multiple languages would create multilingual datasets. Additionally, exploring methods to mitigate template bias and enhance the authenticity of problem statements could improve the utility of the dataset. Conducting studies to assess the quality and educational value of the generated problems through human evaluation would provide valuable insights.

References

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W Ayers, Dragomir Radev, and Jeremy Avigad. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics. *arXiv preprint arXiv:2302.12433*, 2023. 1, 6

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 1, 6

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022. 1

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. 1

Steven Y Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. A survey of data augmentation approaches for nlp. *arXiv preprint arXiv:2105.03075*, 2021. 6

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021. 1, 6

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597, 2015. 6

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022. 6

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*, 2017. 6

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. Openwebmath: An open dataset of high-quality mathematical web text. *arXiv preprint arXiv:2310.06786*, 2023. 1, 6

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 1, 6

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Kang Liu, and Jun Zhao. Large language models are better reasoners with self-verification. *CoRR, abs/2212.09561*, 2023. 6

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023. 6

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mammoth: Building math generalist models through hybrid instruction tuning. *arXiv preprint arXiv:2309.05653*, 2023. 6