TRAINING AND EVALUATING LANGUAGE MODELS WITH TEMPLATE-BASED DATA GENERATION

Yifan Zhang
University of California, Los Angeles
yifzhang@ucla.edu

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 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.

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.

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

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:

- 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. 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.
- We develop TemplateGSM, a dataset comprising over 7 million synthetically generated math problems with verified solutions, addressing the scarcity of large-scale mathematical datasets.
- 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

Template-based Data Generation (TDG) is a systematic approach for producing large volumes of mathematical problems and corresponding solutions using parameterized templates. To advance data augmentation beyond traditional methods, we employ GPT-4 to generate *meta-templates*, broadly-defined structures capturing diverse problem types and linguistic styles. By varying parameters within these GPT-4-generated templates, TDG ensures scalability and maintains high-quality outputs, ultimately supporting the training and evaluation of large language models in mathematical reasoning tasks.

2.1 Methodology

The TDG process comprises several key components that collectively yield a high-quality mathematical dataset:

2.1.1 Generation of Meta-Templates with LLMs

We begin by leveraging large language models (LLMs), such as GPT-4, to generate meta-templates (Zhang et al., 2023). These meta-templates encode the underlying structures of various mathematical problem categories. GPT-4's advanced language-generation capabilities allow us to produce a wide range of templates, each reflecting different mathematical concepts and problem types. Placeholders for variable components (e.g., names, items, quantities, dates, locations) ensure that the final dataset is both linguistically diverse and contextually rich, contributing to a comprehensive training corpus.

2.1.2 Simultaneous Q&A Generation and Verification

In a single integrated step, we generate specific problems and their corresponding solutions by substituting parameters into the GPT-4-produced meta-templates. Each parameter is carefully selected to satisfy certain problem requirements, ensuring both solvability and validity. The simultaneous generation of question and answer pairs preserves consistency between the problem statements and their solutions.

To guarantee correctness and reliability, we employ a **rejection-sampling-based verification** process. This entails running code-based solutions using a code executor and invoking LLMs for solution verification. Any problem-solution pair that fails—due to issues like runtime errors, incorrect results, or ill-formed language—is discarded. Only verified, executable templates are included in the final dataset, safeguarding data integrity.

This iterative generation-and-verification pipeline continues until an adequate number of high-quality problem-solution pairs is produced. By integrating both steps into one workflow, we streamline the data creation process and enhance overall efficiency.

2.2 Process Flowchart

An illustrative overview of our TDG method is shown in Figure 1. The flowchart begins with meta-template generation by an LLM (e.g., GPT-4) and concludes with the final dataset. After

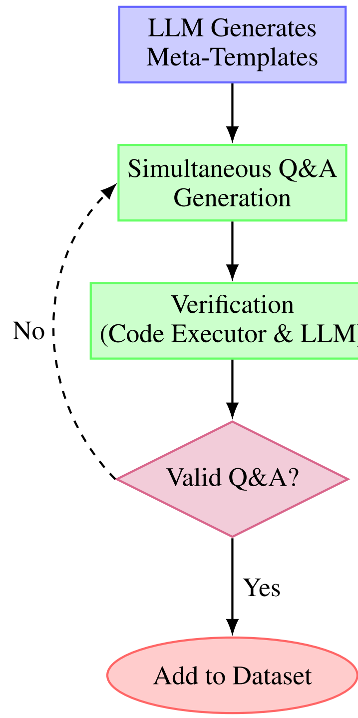

Figure 1: Flowchart illustrating the Template-based Data Generation (TDG) process. An LLM generates meta-templates, which are instantiated into Q&A pairs. These pairs undergo verification, and only the valid ones are added to the dataset. The process loops until the dataset is sufficiently populated.

meta-templates are created, parameters are substituted to instantiate both problems and solutions. The resulting Q&A pairs then undergo verification via code execution and LLM-based checks. Any invalid pairs are discarded, and the process iterates until the dataset is populated with validated, high-quality data.

2.3 CODE IMPLEMENTATION EXAMPLE

An illustrative example of our TDG method is presented in Figure 2. The code snippet shows how we generate problems concerning sales over two consecutive months. The meta-template for this scenario, produced by GPT-4, can be varied through parameter substitution to yield many distinct problems. We maintain lists of random terms (e.g., names, items, months, locations) to ensure diversity and realism. This variation prevents repetition and helps models generalize better.

2.4 GENERATED PROBLEM AND SOLUTION EXAMPLE

To demonstrate TDG's output, consider this example generated from a GPT-4-created template:

Problem:

Emily has 15 apples. She buys 3 times more apples and then gives away 5 apples to her friend. How many apples does Emily have now?

Solution:

Emily starts with 15 apples. She purchases 3 times more, meaning $15 \times 3 = 45$ apples. This brings her total to $15 + 45 = 60$. After giving away 5 apples, she is left with $60 - 5 = 55$. Thus, Emily has 55 apples remaining.

Code-based Solution:

```
# Initial number of apples Emily has
initial_apples = 15
# Emily buys 3 times more apples
```

```python
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 2: An example of our 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.

```python
apples_bought = initial_apples * 3

# Total apples after buying more

total_apples = initial_apples + apples_bought

# Emily gives away 5 apples

apples_given_away = 5

# Apples Emily has now

apples_now = total_apples - apples_given_away

result = apples_now  # Emily has 55 apples now
```

This Q&A pair is generated and verified in a single step, with correctness confirmed via both code execution and LLM checks.

2.5 Advantages of TDG

TDG delivers several notable advantages for generating large-scale mathematical datasets:

- Through parameter variation in GPT-4-generated templates, TDG can produce effectively infinite data, addressing the extensive training needs of large language models.
- By integrating generation and verification into one process, and through code execution and LLM-based validation, every problem-solution pair is confirmed to be correct and consistent. This meticulous supervision boosts dataset quality and model performance.
- GPT-4-generated meta-templates encompass a wide range of problem structures and linguistic variants, improving generalization to unseen problem types, and enabling the synthesis of large-scale, high-quality data.

3 Related Work

Mathematical Datasets. The development of mathematical datasets has been pivotal 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, which filters 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 the structured problem-solution pairs necessary for supervised learning. Similarly, Zhang et al. (2024) proposed using language models as generative verifiers for data selection and released the AutoMathText dataset, but the data is still in unstructured text format.

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-tune models on curated question-answer pairs but are constrained by the availability of high-quality datasets.

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), question rephrasing (Yu et al., 2023), and iterative question composing (IQC) (Liu et al., 2024) 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 precise solution verification through code execution.

4 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. 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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

Haoxiong Liu, Yifan Zhang, Yifan Luo, and Andrew Chi-Chih Yao. Augmenting math word problems via iterative question composing. *arXiv preprint arXiv:2401.09003*, 2024.

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.

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.

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.

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.

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.

Yifan Zhang, Yang Yuan, and Andrew Chi-Chih Yao. Meta prompting for ai systems. *arXiv preprint arXiv:2311.11482*, 2023.

Yifan Zhang, Yifan Luo, Yang Yuan, and Andrew Chi-Chih Yao. Autonomous data selection with language models for mathematical texts. *arXiv preprint arXiv:2402.07625*, 2024.