INVESTIGATING EQUATION-ONLY REASONING IN LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

While Large Language Models excel at mathematical reasoning with Chain-of-Thought prompting, their ability to perform systematic arithmetic reasoning without natural language scaffolding remains poorly understood. We investigate **equation-only supervision**, where LLMs map natural language problems directly to symbolic equation sequences without intermediate explanations. This approach separates reasoning structure generation from arithmetic computation, enabling compact equation storage and deterministic evaluation by external symbolic systems.

We fine-tune LLaMA 3.1 Instruct 8B on GSM8K across three representations: numeric ($16 - 3 - 4 = 9$), symbolic ($v_0 - v_1 - v_2 = v_3$), and semantic variables (eggs_laid_per_day - eggs_eaten_for_breakfast = eggs_sold_at_market). Numeric equations achieve 67.85% accuracy on GSM8K with strong generalization (63.68% on GSM-Symbolic), while semantic variables perform comparably (66.41%). Surprisingly, pure symbolic variables underperform significantly (52.46%), revealing that semantic grounding is crucial for learning equation structures. Our dual evaluation metrics show equation-calculated accuracy often matches or exceeds LLM-calculated accuracy, indicating that improving structure generation—not arithmetic computation—remains the primary challenge. This diagnostic study provides empirical insights into LLMs' structured mathematical reasoning capabilities with implications for building reliable systems leveraging symbolic computation.

1 INTRODUCTION

LLMs have demonstrated impressive capabilities in mathematical reasoning through Chain-of-Thought prompting (Wei et al., 2022), with models like LLaMA 3.1 Instruct 8B achieving 84.5% accuracy on GSM8K, yet their ability to perform systematic arithmetic reasoning without natural language scaffolding remains poorly understood. Math word problems require both extracting mathematical relationships from text and computing correct answers. Current approaches conflate these capabilities by generating natural language explanations alongside mathematical steps.

This work investigates **equation-only supervision**—where models learn to generate symbolic equation sequences without intermediate natural language explanations. We aim to understand whether valid mathematical reasoning structures can be learned from equations alone, and what this reveals about the coupling between structure generation and arithmetic computation in LLMs.

Why equation-only supervision? Equation-only supervision transforms math word problems into an equation extraction problem, where the task is to extract the correct mathematical operations from textual descriptions. While traditional approaches typically only verify the final answer, extracted equations enable programmatic verification of every reasoning step. We can also substitute variables with different numerical values and recalculate outputs using the same equation with no LLMs required. This automated, granular verification cleanly separates the structural reasoning capability (generating correct equation sequences) from arithmetic computation (calculating numerical results), enabling more rigorous analysis of what models learn.

1.1 Problem Formulation and Research Objectives

Consider this GSM8K problem: *"Weng earns $12 an hour for babysitting. Yesterday, she just did 50 minutes of babysitting. How much did she earn?"* Standard Chain-of-Thought produces: *"Weng earns 12/60 = $0.2 per minute. Working 50 minutes, she earned 0.2 × 50 = $10."*

Our equation-only approaches converted the standard chain-of-thought answer to only the mathematical structure in three representations:

1. **Numeric:** $12/60 = 0.2, 0.2 \times 50 = 10$
2. **Symbolic:** $v_0/v_1 = v_2, v_2 \times v_3 = v_4$ with $v_0 = 12, v_1 = 60, v_2 = 0.2, v_3 = 50, v_4 = 10$
3. **Semantic:**

```
hourly_rate/minutes_per_hour = rate_per_minute
rate_per_minute * minutes_worked = total_earnings
```

These three representations serve complementary purposes: The **numeric** format directly evaluates to the answer and tests whether models can extract and sequence mathematical operations correctly. The **symbolic** format abstracts away specific values to focus purely on mathematical structure, testing whether models learn generalizable equation templates independent of concrete numbers. The **semantic** format uses meaningful variable names to represent the conceptual relationships in the problem, bridging abstract mathematical structure with domain semantics and enabling interpretability of the reasoning process.

Research question: Can LLMs learn to generate correct equation structures when trained to produce equation-only outputs, and how does the choice of representation affect this capability?

Our contributions:

- We construct equation-only datasets in three representations (Numeric, Symbolic, Semantic) from GSM8K (Section 3).
- We fine-tune LLaMA 3.1 Instruct 8B on these representations and introduce dual evaluation metrics: *equation-calculated accuracy* (programmatic evaluation) and *LLM-calculated accuracy* (model-generated answers), separating structure generation from arithmetic computation (Section 4).
- We analyze performance across representations, revealing insights about symbolic reasoning capabilities and the trade-offs between different abstraction levels (Section 5).

2 Related Work

Math word problems require parsing text and constructing mathematical operations to solve. Recent work reveals fundamental limitations in LLMs' mathematical reasoning capabilities. Mirzadeh et al. (2024) shows LLMs exhibit noticeable performance variance when only numerical values change, with up to 65% drops when adding clauses, suggesting pattern replication rather than genuine reasoning. Yu et al. (2024) uses extractable symbolic programs to evaluate reasoning, finding that while LLMs generate correct answers, their extracted reasoning fails on problem variations. Deng et al. (2024) shows LLMs struggle with subgroup complexity in arithmetic tasks.

Equation-only supervision offers an alternative approach to mathematical reasoning. Gaur & Saunshi (2023) explores symbolic representations in math word problems, using self-prompting to improve symbolic accuracy by leveraging numeric solutions as concise explanations. Kim et al. (2024) demonstrate that small language models can effectively learn arithmetic reasoning through equation-only formats, achieving strong performance by reducing natural language ambiguity.

Our work extends this paradigm by investigating equation-only supervision across multiple representations (numeric, symbolic, semantic) with dual evaluation metrics that separate structure generation from arithmetic computation, enabling deeper analysis of LLMs' reasoning capabilities.

3 DATASET

We construct our equation-only datasets from the GSM8K benchmark (Cobbe et al., 2021) in the three representations defined in Section 1.1: **Numeric**, **Symbolic**, and **Semantic**.

We process GSM8K's 7,473 training and 1,319 test examples through a multi-stage generation pipeline including regex extraction, LLM-assisted ambiguity resolution, and SymPy verification (detailed in Appendix B). Final processed data size: train 7310/7473 (97.82%), test 1283/1319 (97.27%) (regex-only extraction: train 6282/7473 (84%), test 1065/1319 (80.7%)).

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Model and Training. We fine-tune LLaMA 3.1 Instruct 8B using QLoRA (Dettmers et al., 2023) for parameter-efficient fine-tuning. We used 8-shot Chain-of-Thought prompting with exact match majority voting at temperature 1 (em_maj1@1). Models are trained on equation-only datasets in three representations: numeric, symbolic, and semantic variables.

Evaluation Metrics. We employ two evaluation approaches:

1. **LLM-Generated Results:** Similar to standard GSM8K evaluation, we rely on the LLM to generate the final answer directly, ignoring the intermediate equation chain. This measures end-to-end performance comparable to traditional benchmarks.
2. **Equation-Calculated Answer:** We ignore the LLM's final answer and instead programmatically evaluate the generated equation chain to compute the result. This new metric isolates the model's ability to generate correct reasoning structures independently of its arithmetic computation capabilities.

4.2 GENERALIZATION TESTING

We evaluate generalization by testing fine-tuned models on two benchmarks: (1) the GSM8K test set to measure performance on the standard evaluation distribution, and (2) the GSM-Symbolic benchmark (Mirzadeh et al., 2024), which generates diverse variations of GSM8K problems by modifying numerical values and problem contexts while preserving mathematical structure. This tests whether equation-only trained models can generalize beyond their training distribution to problems with different surface forms but equivalent mathematical reasoning patterns.

5 RESULTS

We evaluate models fine-tuned on equation-only datasets across three representations: numeric, symbolic, and semantic variables. Table 1 presents performance on GSM8K and GSM-Symbolic test sets.

Numeric and semantic representations succeed; symbolic fails. Numeric equations (67.85%) and semantic variables (66.41%) achieve comparable performance on GSM8K, while pure symbolic variables significantly underperform (52.46%), suggesting semantic grounding is crucial for learning equation structures.

Equation-calculated accuracy often matches or exceeds LLM accuracy. This indicates models generate structurally valid equations—failures stem from incorrect structure generation rather than arithmetic errors.

Table 1: Performance comparison across equation representations. **Eval**: LLM-calculated accuracy. **Symbolic Eval**: Equation-calculated accuracy (equations evaluated programmatically).

Training Data	GSM8K		GSM-Symbolic	
	Eval	Sym. Eval	Eval	Sym. Eval
Base model	83.24	-	76.90	-
Standard fine-tuning	76.95	-	72.70	-
Numeric equations	67.85	67.63	63.68	65.70
Symbolic variables	52.46	58.91	46.66	57.84
Semantic variables	66.41	60.50	62.54	64.00

6 Discussion and Conclusion

Our investigation reveals four key findings:

- **Structure generation, not arithmetic, is the bottleneck.** Equation-calculated accuracy often matches or exceeds LLM-calculated accuracy, indicating models successfully generate structurally valid equations. Failures stem from incorrect structure generation rather than computational mistakes.
- **Semantic grounding is crucial.** Numeric (67.85%) and semantic (66.41%) representations enable effective learning, while pure symbolic abstraction (52.46%) significantly underperforms—a 15+ point gap demonstrating that abstraction without grounding hinders learning.
- **Natural language scaffolding has value.** The 17 point drop from baseline (84.5%) to equation-only training (67.85%) quantifies the cost of removing explanations, though maintained equation-calculated accuracy shows models can learn valid reasoning structures from equations alone.
- **Strong generalization to distribution shifts.** Performance on GSM-Symbolic (numeric: 63.68%, semantic: 62.54%) with only 3-4 point drops demonstrates equation-only training produces robust structures beyond pattern matching.

6.1 Future Work

Our findings open several promising research directions: (1) investigating why symbolic abstraction fails and whether modified symbolic representations (e.g., partially grounded variables) could bridge the performance gap, (2) exploring curriculum learning that progressively abstracts from numeric to symbolic representations, (3) scaling equation-only supervision to complex mathematical domains beyond elementary arithmetic, (4) studying whether larger models show better separation between reasoning and computation capabilities, and (5) developing hybrid approaches that combine equation-only supervision with natural language explanations to leverage benefits of both paradigms.

6.2 Conclusion

We investigate equation-only supervision for mathematical reasoning by fine-tuning LLaMA 3.1 Instruct 8B on numeric, symbolic, and semantic representations. Key findings: (1) Models achieve 67.85% accuracy with numeric equations, showing LLMs can learn valid structures without natural language scaffolding, (2) equation structure generation (not arithmetic computation) is the primary challenge (by programmatically evaluating generated equation chains, we find equation-calculated accuracy often exceeds LLM-calculated accuracy), (3) semantic grounding is crucial (pure symbolic abstraction significantly underperforms at 52.46%), and (4) strong generalization to GSM-Symbolic validates robust reasoning beyond pattern matching. These findings advance understanding of LLMs' capabilities in structured mathematical reasoning with implications for symbolic computation systems.

REFERENCES

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.

Lei Deng, Yiming Zhang, et al. Symbolic learning in large language models: Evidence from arithmetic tasks. *arXiv preprint*, 2024.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

Vedant Gaur and Nikunj Saunshi. Reasoning in large language models through symbolic math word problems. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 5889–5903, 2023.

Bumjun Kim, Kunha Lee, Juyeon Kim, and Sangam Lee. Small language models are equation reasoners. *arXiv preprint arXiv:2409.12393*, 2024.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

Xiaodong Yu, Ben Zhou, Hao Cheng, and Dan Roth. Reasonagain: Using extractable symbolic programs to evaluate mathematical reasoning. *arXiv preprint arXiv:2410.19056*, 2024.

A DATASET EXAMPLE

This section provides a complete example showing how we extract equation-only representations from GSM8K problems.

A.1 ORIGINAL GSM8K PROBLEM

Question: Betty is saving money for a new wallet which costs $100. Betty has only half of the money she needs. Her parents decided to give her $15 for that purpose, and her grandparents twice as much as her parents. How much more money does Betty need to buy the wallet?

Answer: In the beginning, Betty has only 100 / 2 = $<<100/2=50>>50. Betty's grandparents gave her 15 * 2 = $<<15*2=30>>30. This means, Betty needs 100 - 50 - 30 - 15 = $<<100-50-30-15=5>>5 more. 5

Answer Value: 5

A.2 EXTRACTED REPRESENTATIONS

A.2.1 NUMERIC EQUATIONS

```
100/2 = 50
15*2 = 30
100-50-30-15 = 5
```

A.2.2 SYMBOLIC REPRESENTATION

Symbolic Equations:

```
v0/v1 = v2
v3*v1 = v4
v0-v2-v4-v3 = v5
```

Variable Mapping:

```
v0 = 100.0    v1 = 2.0      v2 = 50.0
v3 = 15.0     v4 = 30.0     v5 = 5.0
```

A.2.3 Semantic Variable Representation

Semantic Variable Equations:

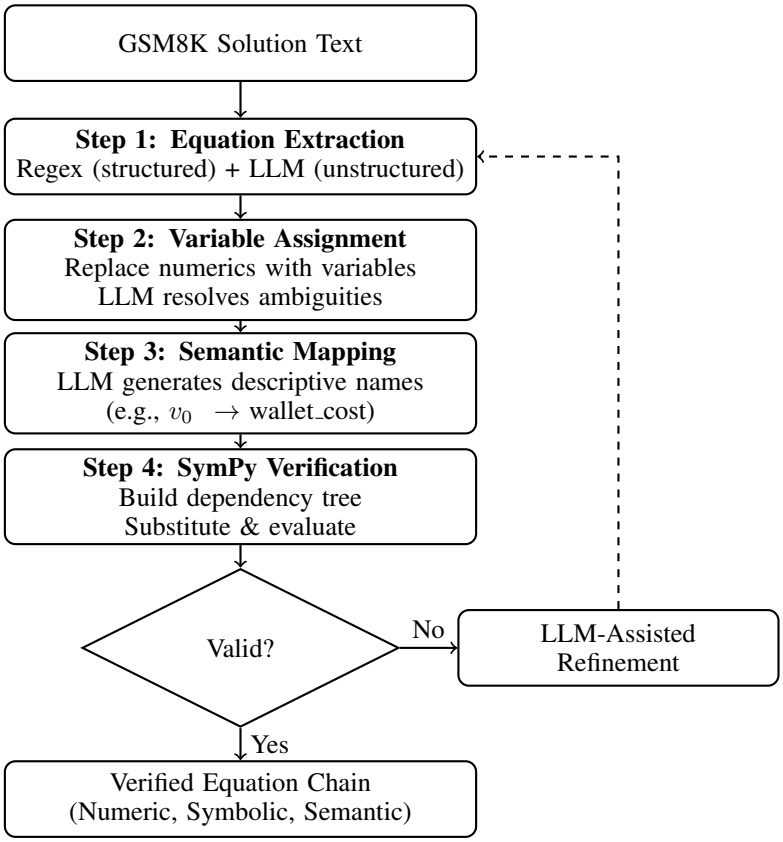
```
wallet_cost/divisor_two = betty_initial_money
parents_gift*divisor_two = grandparents_gift
wallet_cost-betty_initial_money-grandparents_gift-parents_gift
  = money_still_needed
```

Semantic Variable Mapping:

```
wallet_cost = 100.0
divisor_two = 2.0
betty_initial_money = 50.0
parents_gift = 15.0
grandparents_gift = 30.0
money_still_needed = 5.0
```

B Data generation

B.1 Multi-Stage Pipeline



Step 1: Equation extraction. We parse GSM8K solutions to identify equation patterns embedded in the solution text. For structured data, we use regular expressions to extract equations (e.g.,

the pattern <<18*0.5=9.00>>9.00 is extracted as ["18*0.5=9.00"]). For noisy or unstructured solution formats, including entries without explicit <<...>> delimiters, we employ LLM-based extraction to handle variability in equation presentation. The LLM identifies mathematical expressions embedded in natural language text and extracts them in a structured format, even when equation boundaries are not explicitly marked.

Step 2: Automated variable assignment with LLM-assisted ambiguity resolution. We replace numeric values with variables to create symbolic representations. A critical challenge arises from **variable mapping ambiguity**: when the same numeric value appears in different semantic contexts. To resolve these ambiguities, we use an LLM to analyze problem context and assign unique variables to semantically distinct values, even when numerically identical.

Step 3: LLM-based semantic mappings. For the semantic variable representation, we employ an LLM to generate meaningful variable names that capture the semantic role of each quantity in the problem. The LLM analyzes the original problem text and assigns descriptive names (e.g., wallet_cost, betty_initial_money) rather than generic placeholders (e.g., v_0, v_1). This step produces human-interpretable equation chains that preserve both mathematical structure and semantic meaning.

Step 4: Symbolic verification with SymPy. We verify the correctness of all extracted equation chains using the SymPy symbolic mathematics library. The verification process includes: (1) building a dependency tree by parsing equations to identify dependencies, (2) recursively substituting intermediate variables starting from the final equation (e.g., $v_0 + v_2 = v_3$ becomes $v_0 + v_0 \cdot v_1 = v_3$ after substituting v_2), (3) evaluating the resolved expression using SymPy with variable→value mappings, and (4) verifying the computed result matches the expected answer within a tolerance of 10^{-6}. This step detects circular dependencies, parse errors, evaluation errors, and validation failures. Invalid equation chains are flagged and iteratively corrected using LLM-assisted refinement.

B.2 LLM Prompts

Equation extraction prompt:

```
Extract all calculation steps from the following answer text and convert them to properly
    ↪ formatted equations.

Answer text:
{answer_text}

Expected final answer: {answer_value}

Previous extraction:
equations: {equations}
variable_equations: {variable_equations}
variable_mapping: {variable_mapping}
validation_error: {validation_error}

Instructions:
1. Extract ALL equations from the text (marked with <<...>>) even if calculations appear
    ↪ incorrect
2. Convert ALL numbers to variables (v0, v1, v2, etc.) in variable_equations
3. IMPORTANT: Replace decimals starting with a dot (like .5) with proper format (0.5) in
    ↪ equations
4. Each unique number gets its own variable, numbers used multiple times get the same
    ↪ variable
5. Output ONLY a JSON code block in the format shown below (no other text)

Output format:
```json
{{
   "equations": ["equation1", "equation2", ...],
   "variable_equations": ["v_equation1", "v_equation2", ...],
   "variable_mapping": {{"v0": "value0", "v1": "value1", ...}},
   "corrected_answer": "correct_final_answer" (optional, only if calculations in text are
       ↪ wrong)
}}
```

Example 1 (correct calculations):
Input text:
She calculates 18*.5 = <<18*0.5=9.00>>9.00 for overtime rate.
Her pay is 18+9 = <<18+9=27.00>>27.00.
27
```

```
Output:
```json
{{
   "equations": ["18*0.5=9.00", "18+9=27.00"],
   "variable_equations": ["v0*v1=v2", "v0+v2=v3"],
   "variable_mapping": {{"v0": "18", "v1": "0.5", "v2": "9.00", "v3": "27.00"}}
}}
```

Example 2 (incorrect calculations in text):
Input text:
He has 10+5 = <<10+5=20>>20 apples.
He gives away 20-3 = <<20-3=17>>17.
17

Output:
```json
{{
   "equations": ["10+5=20", "20-3=17"],
   "variable_equations": ["v0+v1=v2", "v2-v3=v4"],
   "variable_mapping": {{"v0": "10", "v1": "5", "v2": "20", "v3": "3", "v4": "17"}},
   "corrected_answer": "12"
}}
```
```

Semantic variable naming prompt:

```
Given a math problem and its variables, rename the generic variables (v0, v1, v2, etc.)
    ↪ to descriptive, contextually meaningful names.

Question: {question_text}

Answer text: {answer_text}

Current variables and their values:
{json.dumps(variable_mapping, indent=2)}

Current equations:
{json.dumps(variable_equations, indent=2)}

Instructions:
1. Analyze the question and answer to understand what each variable represents
2. Create descriptive variable names using snake_case (e.g., number_of_apples,
    ↪ price_per_item, total_cost)
3. Be specific and clear - avoid generic names like "value", "number", "result"
4. Keep names concise but descriptive (prefer "hours_worked" over "h" or "
    ↪ the_number_of_hours_that_were_worked")
5. For intermediate calculations, use names like "subtotal", "partial_sum", "
    ↪ intermediate_result" if the context is clear
6. Output ONLY a JSON object mapping old names to new names (no other text)

Output format:
```json
{{
   "v0": "descriptive_name_1",
   "v1": "descriptive_name_2",
   "v2": "descriptive_name_3"
}}
```

Example:
Question: "John has 10 apples. He buys 5 more. How many does he have?"
Variables: {{"v0": "10", "v1": "5", "v2": "15"}}
Equations: ["v0+v1=v2"]

Output:
```json
{{
   "v0": "initial_apples",
   "v1": "apples_bought",
   "v2": "total_apples"
}}
```
```