Direct Induction Proof Challenge: Evaluating Large Language Models on Deeply Nested Mathematical Induction

Risako Ando [1] **Koji Mineshima** [1] **Mitsuhiro Okada** [1]

Abstract

We introduce a challenge designed to evaluate the capability of Large Language Models (LLMs) in performing mathematical induction proofs, with a particular focus on nested induction. Our task requires models to construct direct induction proofs in both formal and informal settings, without relying on any preexisting lemmas. Experimental results indicate that current models struggle with generating direct induction proofs, suggesting that there remains significant room for improvement.

1. Introduction

It has become widely recognized in recent years that LLMs possess a certain capacity for logical reasoning and its arithmetical extensions, including the ability to construct proofs (Lightman et al., 2024). However, many of these successful instances might be attributable to the reuse and combination of lemmas contained in the training data. This corresponds to the use of pre-existing lemmas in libraries of automated theorem proving and proof assistant systems.

In this study, we focus on proof construction within the domain of elementary part of arithmetic, under conditions that restrict access to such lemmas. Specifically, we investigate the extent to which LLMs can construct proofs involving mathematical induction of varying complexity, using the depth of nested induction as a measure of complexity, in contrast to the complexity of formulas (Dean & Naibo, 2024). Furthermore, we evaluate whether LLMs can generalize to more complex induction proofs when given examples of simpler ones in a few-shot prompting setting.

2. Related Work

Research on automating mathematical induction dates back to the 1970s (Boyer & Moore, 1979), with foundational systems such as the Boyer-Moore theorem prover establishing early techniques for mechanized reasoning.

Modern proof assistants such as Coq and Lean provide robust support for inductive types and user-guided inductive reasoning. However, despite these advancements, the full automation of inductive proofs remains a significant challenge. In practice, users are still required to manually design induction schemas or supply key lemmas, and general-purpose automation often fails to scale beyond relatively simple proofs.

Recently, several benchmark suites have been proposed to evaluate inductive theorem provers (Claessen et al., 2015; Hajdú et al., 2021; Gauthier et al., 2023). While these datasets have advanced the evaluation of inductive provers, very few focus on deeply nested or multi-level induction, which remains a major challenge for both automated and neural proof systems.

Recent work on large language models (LLMs) has shifted focus from answer-only evaluation to step-wise verification, achieving impressive results on math benchmarks via process supervision (Uesato et al., 2022; Lightman et al., 2024). However, research specifically addressing mathematical induction is still scarce.

Dean & Naibo (2024) provide a broader critique of LLM-based mathematical reasoning, classifying theorems by arithmetical complexity and highlighting key limitations in current models (Romera-Paredes et al., 2024). In contrast, our work focuses explicitly on mathematical induction—an area where even symbolic provers struggle—and investigates whether LLMs can construct valid induction proofs in formal contexts.

3. Direct induction Proof Challenge

LLMs tend to employ induction in a shallow manner, often relying on existing libraries of lemmas. To investigate how well LLMs can support proofs involving nested induction, we focus on what we call *direct induction proofs*. Suppose an inductive data type is given, along with a set of primitive recursive function definitions induced from it. When the structural induction principle is provided as an inference rule, we define a direct induction proof as a proof of a uni-

[1]Department of Philosophy, Keio University, Tokyo, Japan. Correspondence to: Risako Ando <risakochaan@keio.jp>, Koji Mineshima <minesima@abelard.flet.keio.ac.jp>.

The second AI for MATH Workshop at the 42nd International Conference on Machine Learning, Vancouver, Canada.

versally quantified equation that is constructed solely from the given definitions and the structural induction principle—that is, without invoking any auxiliary lemmas. We assume equational logic as the background logic. An example of direct induction proof is given in Appendix A.1.

Among various inductive data types, a fundamental case is that of the natural numbers. Primitive Recursive Arithmetic (PRA) (Skolem, 1967) corresponds to this setting, where the inductive type is the natural numbers. For the purposes of this study, we primarily focus on a fragment of PRA in which addition and multiplication are defined directly. This fragment is sufficiently rich in that it allows multiply nested induction, making it a valuable first step for examining the automation of direct induction proofs.

4. Experiments

4.1. Problem Set

To evaluate the capabilities of LLMs in constructing direct induction proofs, we designed a benchmark consisting of 20 arithmetic statements involving addition and multiplication. These problems were selected to cover a spectrum of increasing structural complexity in terms of the number of variables and the depth of induction required. See Appendix A.2 for the full list of statements and variable counts.

4.2. Proof Generation Tasks

For each problem, we instruct LLMs to generate two types of proofs: informal proofs in natural language (English) and formal proofs written in Lean 4 (`https://lean-lang.org/`). For informal proofs, we manually examine the correctness of proofs. For formal proofs, we use automatic evaluation with Lean 4.

Informal Proof We conduct experiments on informal proofs under two settings. First, in the **Direct** task, the use of any lemmas is strictly prohibited. The model is required to rely solely on the definitions of addition and multiplication, along with mathematical induction on natural numbers (including nested induction when necessary). Prompt examples are provided in Appendix A.3. Second, we test the **Lemma** task, where the use of lemmas is permitted, but the model is required to provide a proof for each lemma it uses. For both the **Direct** and **Lemma** tasks, we provided two-shot examples. These included a proof of the statement $a + \mathrm{succ}(0) = \mathrm{succ}(a)$, and a proof of $a + b = b + a$ involving double induction (See Appendix A.1).

Formal Proof For formal proofs, in addition to the **Direct** and **Lemma** tasks, we test a third task called the **Library** task. In this setting, the use of lemmas from external sources such as Lean's Mathlib is allowed, but automated proof tactics are strictly prohibited. In the formal proof setting, we implemented a feedback loop based on Lean 4's error messages. The model was allowed to retry up to n times (with $n = 1, 5, 10$) in an attempt to generate a correct proof.

Experimental Setting We use several large language models of varying sizes in our experiment: openai/gpt-4o-2024-08-06, openai/gpt-3.5-turbo-0125, and meta-llama/llama-3-70b-instruct. We set the temperature to 0.2 and the maximum number of output tokens to 2000, while leaving other parameters at their default values.

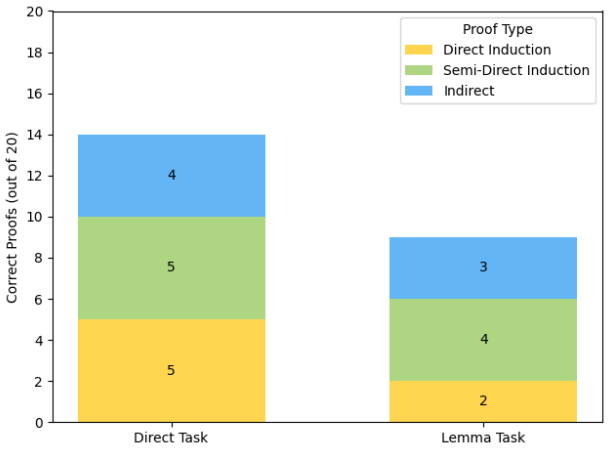

Figure 1. The results of gpt-4o on each task in informal proofs. The numbers indicate the number of correct proofs out of a total of 20 problems.

4.3. Informal Proof Results

Figure 1 shows the results of the **Direct** and **Lemma** tasks in informal proofs. Given the high cost of manual evaluation, we limited our human assessment to the best-performing model (gpt-4o). A common error pattern observed was the failure to adhere to the provided definitions (see Appendix A.4, with proofs including unproven steps such as $a + \mathrm{succ}(a) = \mathrm{succ}(a + b)$ (right addition), even though our definition is $\mathrm{succ}(a) + b = \mathrm{succ}(a + b)$ (left addition). Thus, we relax the condition and evaluate the results according to the three criteria: (1) **Direct Induction Proof**: Correct as a direct induction proof, where no step other than mathematical induction and definitions appears; (2) **Semi-Direct Induction Proof**: A direct proof except that it uses the right addition or multiplication rules without proving them by induction; (3) **Indirect Proof**: Neither direct nor semi-direct but the proof is correct. Any proof that does not fall under these three categories was considered incorrect.

When the evaluation criteria were relaxed to include semi-direct induction proofs and indirect proofs, the model correctly answered 14 out of 20 problems. However, it achieved only 5 correct answers out of 20 under the strict direct proof

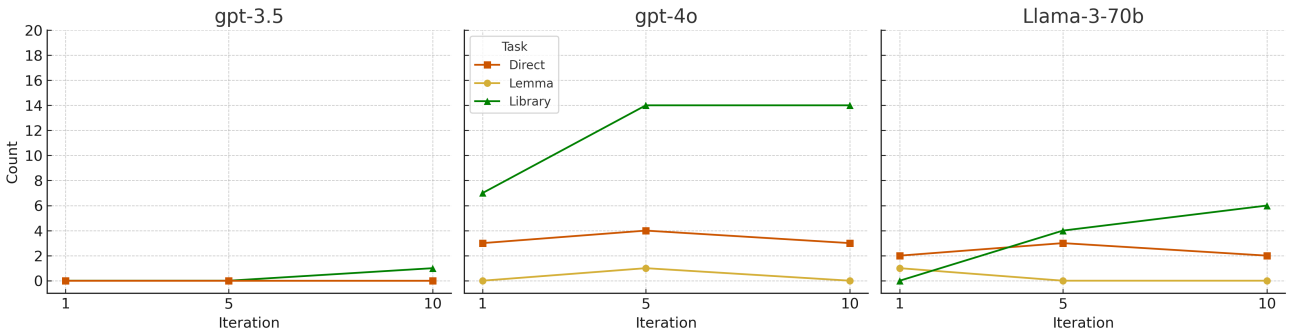

Figure 2. The results of each task in formal proofs. **Iteration** refers to the number of times the error-feedback process was attempted. **Count** refers to the number of correctly generated proofs out of 20 problems.

setting. Performance was poorer on the **Lemma** task, which requires the model to formulate and prove auxiliary lemmas, providing detailed justification for each step. This suggests that current models struggle with generating such explanatory intermediate steps.

Regarding the generalization abilities of LLMs in theorem proving, the following results were observed: (1) Despite not being shown any examples involving multiplication, the model was occasionally able to construct correct direct induction proofs involving multiplication, indicating a degree of generalization beyond addition (ID 8 in Appendix A.2). (2) Although only examples involving double induction were provided, the model was able to correctly prove a theorem requiring triple nested induction (ID 12 in Appendix A.2).

4.4. Formal Proof Results

Figure 2 shows the results of the three tasks in formal proofs. The models showed poor performance in constructing direct induction proofs. Increasing the number of attempts using error feedback did not improve performance on the **Direct** or **Lemma** tasks. Performance on the **Lemma** task—where the model is required to formulate and prove its own auxiliary lemmas—was extremely poor. In contrast, performance improved in the **Library** task, where the model was allowed to refer to existing libraries. Similar to the informal proof setting, the **Lemma** task requires the model to provide detailed explanations for each proof step in the form of lemmas. Again, this suggests that current models are still weak at producing such explanatory reasoning.

5. Summary and Future Work

We proposed the Direct Induction Proof Challenge, a benchmark designed to evaluate LLMs' ability to construct direct mathematical induction proofs, particularly those involving nested induction, without relying on auxiliary lemmas. Based on a fragment of Primitive Recursive Arithmetic (PRA), we tested models in both informal (natural language) and formal (Lean 4) settings across Direct, Lemma, and Library tasks.

Our results show that while LLMs sometimes generalize beyond the few-shot examples provided in the in-context learning setup (such as successfully applying triple induction or handling multiplication without direct prompts), they still face significant challenges in producing strictly direct proofs and in formulating and justifying intermediate lemmas.

For future work, We plan to extend our experiments within the framework of PRA to encompass functions beyond addition and multiplication. We also intend to broaden our investigation to include more general inductive data types, allowing for an evaluation of mathematical induction in the context of more general forms of structural induction.

References

Boyer, R. S. and Moore, J. S. *A Computational Logic*. Academic Press, New York, 1979.

Claessen, K., Johansson, M., Rosén, D., and Smallbone, N. Tip: Tons of inductive problems. In *Intelligent Computer Mathematics (CICM)*, volume 9150 of *Lecture Notes in Computer Science*, pp. 333–337. Springer, 2015. URL https://github.com/tip-org/benchmarks.

Dean, W. and Naibo, A. Artificial intelligence and inherent mathematical difficulty. *arXiv preprint arXiv:2408.03345*, 2024. URL https://arxiv.org/abs/2408.03345.

Gauthier, T., Brown, C. E., Janota, M., and Urban, J. A mathematical benchmark for inductive theorem provers.

In *Proceedings of 24th International Conference on Logic*, volume 94, pp. 224–237, 2023.

Hajdú, M., Hozzová, P., Kovács, L., Schoisswohl, J., and Voronkov, A. Inductive benchmarks for automated reasoning. In *Intelligent Computer Mathematics (CICM)*, volume 12833 of *Lecture Notes in Computer Science*, pp. 124–129. Springer, 2021.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J., Ellenberg, J. S., Wang, P., Fawzi, O., et al. Mathematical discoveries from program search with large language models. *Nature*, 625 (7995):468–475, 2024.

Skolem, T. The foundation of elementary arithmetic established by means of the recursive mode of thought, without the use of apparent variables ranging over infinite domains. In van Heijenoort, J. (ed.), *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*, pp. 302–333. Harvard University Press, 1967. Original work published in 1923.

Uesato, J., Kushman, N., Kumar, R., Song, F., Siegel, N., Wang, L., Creswell, A., Irving, G., and Higgins, I. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

A. Appendix

A.1. Example of direct induction proof involving double-induction provided as few-shot

The following is an example of a direct induction proof provided as few-shot in the informal proof setting.

```
We show that for all natural numbers a and b, the following statement holds:
    a + b = b + a
Now we prove the theorem by induction on a.

Case 1 (Base Case):
We prove 0 + b = b + 0 by induction on b. (Inner induction)

    Case 1.1 (Base Case):
    By definition, 0 + 0 = 0 = 0 + 0

    Case 1-2 (Induction Step):
    Assume induction hypothesis (ih): 0 + b = b + 0
    We show: 0 + s(b) = s(b) + 0

    0 + s(b) = s(b)           (by definition of +)
             = s(0 + b)       (by definition of +)
             = s(b + 0)       (by ih) \\
             = s(b) + 0       (by definition of +)

Case 2 (Induction Step):
Assume induction hypothesis (ih): a + b = b + a.
We show: s(a) + b = b + s(a).

s(a) + b = s(a + b)        (by definition)
         = s(b + a)        (by ih)
         = s(b) + a        (by definition)

We prove s(b) + a = b + s(a) by induction on b. (Inner induction)

    Case 2.1 (Base Case):
    s(0) + a = s(0 + a) = s(a) = 0 + s(a)

    Case 2.2 (Induction Step):
    Assume induction hypothesis (ih'): s(b) + a = b + s(a)
    Goal: s(s(b)) + a = s(b) + s(a)

    s(s(b)) + a = s(s(b) + a)     (by definition)
                = s(b + s(a))     (by ih')
                = s(b) + s(a)     (by definition)

Thus, we have shown that a + b = b + a for all natural numbers a and b.
```

A.2. Problems

The following shows the set of all 20 example problems we used in our experiments. The "Variable" column indicates the number of parameters present in the corresponding statement.

ID	Statement	Variable
1	$a+1=1+a$	1
2	$2+a=a+2$	1
3	$a\times 1=a$	1
4	$1\times b=b\times 1$	1
5	$(a+1)+1=a+(1+1)$	1
6	$a\times b=b\times a$	2
7	$(a+b)+1=a+(b+1)$	2
8	$(a\times 1)\times c=a\times(1\times c)$	2
9	$(a+1)\times c=(a\times c)+(1\times c)$	2
10	$1\times(b+c)=(1\times b)+(1\times c)$	2
11	$(a+b)+c=a+(b+c)$	3
12	$a\times(b+c)=(a\times b)+(a\times c)$	3
13	$(a\times b)\times c=a\times(b\times c)$	3
14	$(a+b)\times c=(a\times c)+(b\times c)$	3
15	$(1+(b+c))\times d=(1\times d)+((b\times d)+(c\times d))$	3
16	$(a+b)\times(c+d)=((a\times c)+(a\times d))+((b\times c)+(b\times d))$	4
17	$((a+b)+c)+d=a+(b+(c+d))$	4
18	$(a\times b)\times(c\times d)=a\times(b\times(c\times d))$	4
19	$(a+b)\times(c\times d)=(a\times(c\times d))+(b\times(c\times d))$	4
20	$((a+b)\times c)+((a+b)\times d)=(a+b)\times(c+d)$	4

A.3. Prompts

The following are prompt examples for the **Direct** Task in formal and informal proorfs.

Prompt example (Direct, Formal)

```
Prove the following statement in Lean 4, adhering to the strict constraints listed below.

{Statement}

    Proof Constraints:
    - Do not use any predefined lemmas, such as add_zero, add_comm, etc.
    - Only use mathematical induction on natural numbers, including nested induction if necessary.
    - Do not use any automated tactics (e.g., simp, ring, omega, or similar).
    - Do not use syntax sugar such as |>.
    - Use Lean 4 syntax and avoid using syntax that only works in Lean 3.
    - Carefully follow the definitions of natural numbers, addition, multiplication as provided below.
    - Use the sample proof below as a model for writing a correct proof.

    Output Format:
    - Provide the complete proof in Lean 4 code block format, starting and ending with triple backticks (```lean```).

{Example}
```

Prompt example (Direct, Informal)

```
Prove the following statement. Your proof must adhere to the following strict constraints:

{Statement}

    - Do not invoke any preestablished lemmas such as the commutativity of addition or multiplication, or similar.
    - Do not assume any unproven propositions such as "a + 0 = 0 for all natural numbers a".
    - Use only mathematical induction on natural numbers, including nested induction if necessary.
    - Adhere strictly to the definitions of natural numbers, addition, multiplication, as given below.
    - Follow the structure of the sample proof provided.

{Example}
```

A.4. Definition

The following is a definition of addition and multiplication used in the prompt.

Definition (Formal)

```
inductive Nat where
  | o : Nat
  | s : Nat → Nat

def add : Nat → Nat → Nat
  | o, b => b
  | s a, b => s (add a b)

def mul : Nat → Nat → Nat
  | o, _ => o
  | s a, b => add (mul a b) b

infixl:60 " + " => add
infixl:70 " * " => mul
```

Definition (Informal)

```
We define natural numbers inductively:
  - 0 is a natural number.
  - If a is a natural number, then so is s(a), the successor of a.

Addition is defined recursively:
  - 0 + b = b
  - s(a) + b = s(a + b)

Multiplication is defined recursively:
  - 0 * b = 0
  - s(a) * b = (a * b) + b
```