Large Language Models for Mathematical Problem Solving: Applications, Challenges and Future Directions

Yuxuan Wan

Department of Applied Mathematics, University of California, Santa Barbara, CA 93106, United States of America

yuxuan_wan@ucsb.edu

Abstract. Large language models (LLMs) have made remarkable strides in natural language tasks, but mathematical problem solving remains a significant challenge. This paper surveys recent methods developed to enhance the mathematical reasoning abilities of LLMs. Techniques such as chain-of-thought prompting, self-consistency decoding, retrieval-based augmentation, and tool use have markedly improved performance on math benchmarks. Despite progress, the LLMs still struggle with reasoning faithfulness, generalization to new problems, and arithmetic accuracy. This paper identifies three key challenges and discuss corresponding future directions, including structured reasoning formats, automatic verification, and targeted training. Bridging the gap between current LLM performance and human-level mathematical proficiency will not only advance AI's capabilities in math but also improve the transparency and reliability of reasoning in AI systems more broadly. Ultimately, strengthening LLMs in mathematics represents a step toward Artificial Intelligence (AI) systems that can provide trustworthy, interpretable, and high-value support in education, scientific research, and other domains requiring precise reasoning.

Keywords: Large Language Model, Mathematical Problem Solving, deep learning.

1. Introduction

Large language models (LLMs) have achieved state-of-the-art performance on a wide range of natural language processing tasks. Scaling up model size into the hundreds of billions of parameters has been a key factor in these successes. However, tasks requiring complex, multi-step reasoning – notably mathematical problem solving – remain challenging even for the most advanced LLMs. Solving math word problems or performing formal logical reasoning pushes LLMs beyond straightforward pattern recognition, since it demands precise step-by-step deduction and error-free calculation. A typical math problem requires the solver to parse a description, recall relevant domain knowledge (e.g. formulas or theorems), and carry out a sequence of symbolic or arithmetic operations to arrive at a correct solution. These requirements highlight a fundamental gap in the reasoning capabilities of current LLMs.

Research into improving LLMs on mathematical tasks is motivated by both scientific and practical goals. Robust reasoning in mathematics is seen as a litmus test for an AI system's ability to handle complex, structured problem solving. Moreover, in educational and scientific domains, an AI that can reliably solve math problems – and explain its solutions – would be immensely valuable. Yet today's AI still struggles with multi-step reasoning that many schoolchildren find routine. In light of this, a growing body of work focuses on methods to enhance the reasoning abilities of LLMs for math problem solving. The following sections review state-of-the-art prompting and training strategies developed for this purpose (Section 2), discuss the current challenges that limit LLM performance on math reasoning, and outline potential future directions to bridge these gaps (Section 3). Finally, Section 4 concludes with a summary of findings and implications.

2. LLMs in mathematical problem solving

2.1. Chain-of-thought prompting

One prominent approach for enabling better mathematical reasoning in LLMs is chain-of-thought (CoT) prompting. In chain-of-thought prompting, the model is guided (through its input prompt) to produce a step-by-step reasoning process before giving the final answer, rather than directly outputting the answer in one step. Concretely, the prompt includes examples of questions paired with detailed sequences of intermediate reasoning steps that lead to the correct answer. By conditioning on such exemplars, the model is encouraged to decompose complex problems into simpler sub-steps, mimicking an intuitive human problem-solving process. This method does not require updating the model's weights – it uses prompt engineering to induce a more logical reasoning behavior.

Chain-of-thought prompting has yielded significant improvements in mathematical problem solving and other reasoning tasks. Importantly, it was found to be an emergent ability that only manifests in sufficiently large models. Models with on the order of 100 billion parameters or more exhibit substantial gains from CoT prompting, whereas smaller models see little benefit. For example, Wei et al. showed that with chain-of-thought prompting, a 540B-parameter model (Google's PaLM) achieved a new state-of-the-art 58% accuracy on the GSM8K benchmark of grade-school math problems [1]. This exceeded the previous best of 55%, which had been obtained by fine-tuning GPT-3 (175B) on a math dataset and using a trained verifier to select answers. In other words, prompting a large language model to "think step by step" allowed it to solve math problems about as well as an explicitly fine-tuned approach, underscoring the power of in-context reasoning. Moreover, chain-of-thought prompting has broad applicability: it not only improves arithmetic word problems, but also helps on logical reasoning and commonsense question-answering tasks by enabling the model to break down queries into interpretable steps.

An important innovation related to CoT prompting is the discovery that explicit few-shot examples are not the only way to induce this behavior. Kojima et al. found that simply appending a phrase like "Let's think step by step." to a question can trigger zero-shot chain-of-thought reasoning in a large language model [2]. This Zero-Shot CoT technique allows the model to generate a coherent reasoning chain on the fly, significantly improving zero-shot performance on reasoning tasks without any hand-crafted examples. For instance, zero-shot CoT prompting enabled GPT-3 to outperform standard prompting (with no reasoning) and even approach the accuracy of few-shot CoT in some cases. The ability to elicit multi-step reasoning in a zero-shot manner highlights that LLMs possess latent reasoning capabilities which can be unlocked with the right prompt. Overall, chain-of-thought prompting – in both few-shot and zero-shot forms – has emerged as a simple yet powerful tool for tackling mathematical problems by encouraging a logical breakdown of the solution process.

2.2. Self-consistency decoding

While chain-of-thought prompting improves reasoning, relying on a single greedy reasoning path can still lead the model astray if it makes an early mistake. To address this, Wang et al. proposed a decoding strategy called self-consistency, which further enhances performance by using an ensemble of reasoning paths [3]. Instead of generating one definitive solution, the model is prompted (with CoT) to sample multiple independent chains-of-thought for the same problem by introducing randomness into the decoding process. After generating, say, 10 or 50 different reasoning chains and their final answers, the most common answer is selected by majority vote. The intuition is that although any single chain might go wrong, the correct answer – if reachable by the model at all – is likely to appear in at least a few of the chains. By marginalizing out the specific path and choosing the answer that most chains agree on, self-consistency aims to identify the model's most likely correct answer.

Empirical results demonstrate that self-consistency decoding produces a substantial jump in accuracy on reasoning benchmarks. Wang et al. reported striking improvements such as a +17–18% absolute accuracy gain on GSM8K math problems, compared to standard CoT prompting with greedy decoding [3]. Using self-consistency, the GSM8K performance of a CoT-prompted 540B model

increased from roughly 57% to 75% – a new state-of-the-art at the time. Similar boosts were observed on other datasets (e.g. an increase of about 12% on the SVAMP arithmetic benchmark). Google's researchers likewise noted that applying a "majority vote" across many reasoning paths raised GSM8K accuracy from 58% to about 74%. Self-consistency works because it leverages the model's own uncertainty: by exploring diverse solution paths, the model's final consensus answer is more reliable than any single trial. This method is model-agnostic and requires no additional training, altering only how the model's outputs are decoded. Self-consistency has thus become a standard technique to maximize reasoning performance, especially in domains like math where one correct answer exists and different solution approaches should converge to the same result. The success of self-consistency underlines the benefit of marrying generation with a validation heuristic – effectively using redundancy and voting to correct the model's reasoning mistakes.

2.3. Other strategies

In addition to prompting methods, researchers have explored various training strategies and model augmentations to improve LLM performance on mathematical tasks. Several prominent approaches are outlined below:

Fine-Tuning on Mathematical Corpora: A straightforward approach is to fine-tune or specialize a pre-trained LLM on domains rich in mathematical content. By exposing the model to more formal mathematics during training, it can learn domain-specific reasoning patterns and vocabulary. A notable example is Google's Minerva model, which was based on a 540B-parameter LLM (PaLM) further trained on a high-quality corpus of scientific papers, math websites, and other technical content. This additional fine-tuning imbued the model with knowledge of formulas and the ability to produce well-formatted solutions (including LaTeX equations). Minerva achieved state-of-the-art performance on several quantitative reasoning benchmarks, including the MATH competition dataset and GSM8K, without using external tools. It was able to correctly answer nearly a third of university-level science and engineering questions, vastly outperforming earlier LLMs on those tasks. Fine-tuning clearly helps models internalize mathematical reasoning. Another fine-tuning technique is the scratchpad or "show your work" approach, where models are trained to generate intermediate calculation steps explicitly before giving the final answer. For instance, even smaller models can learn to output step-by-step solutions when fine-tuned on such data, leading to more accurate results on multi-step arithmetic problems. In practice, the best results often come from combining fine-tuning with advanced prompting or decoding techniques, rather than fine-tuning alone.

Retrieval-Augmented Generation: Pure language models have a fixed store of implicit knowledge in their weights and limited ability to perform exact calculation. Retrieval-augmented generation (RAG) addresses this by equipping the LLM with access to an external knowledge base or tool that it can query in the middle of solving a problem. In a math context, a model might search a database of formulas, theorems, or previously solved questions to fetch relevant information. Recent research on Retrieval-Augmented Thought demonstrates that allowing an LLM to iteratively retrieve text (e.g. from Wikipedia) while building its chain-of-thought can significantly improve reasoning accuracy and reduce hallucinations [4]. In fact, augmenting GPT-3.5/GPT-4 with a targeted retrieval step before each reasoning hop led to about 17% relative improvement on mathematical reasoning benchmarks on average. The intuition is that whenever the model is uncertain or lacking a fact (say, a physics constant or a geometry formula), it can consult an external resource to ground its reasoning. Retrieval augmentation thus extends the model's effective knowledge beyond what it memorized during training, making this approach especially powerful for knowledge-intensive math word problems or open-ended questions.

Tool Use and Programmatic Reasoning: Related to retrieval is the idea of giving LLMs the ability to use computational tools (such as calculators, Python interpreters, or algebra solvers) to offload tasks that are difficult for a neural network. Mathematics is an ideal domain for this, since many problems ultimately reduce to calculations or symbolic manipulations that a computer program can perform exactly. One instantiation of this idea is Program-Aided Language Models (PAL), proposed

by Gao et al. [5]. In PAL, the language model is prompted to output a short computer program (e.g. in Python) that, when executed, produces the answer to the question. For example, given a math word problem, instead of computing the answer entirely in free-form text, the model might generate code to perform the required arithmetic or iterative procedure. Because the code can be run with perfect accuracy, the final answer is more likely to be correct as long as the model's program faithfully represents the problem logic. Empirical results showed that even moderately sized models paired with a Python interpreter can exceed the accuracy of much larger models that work purely in natural language. Similarly, Chen et al. introduced Program-of-Thoughts prompting, which encourages the model to separate computation from reasoning (for instance, by producing pseudo-code for calculations) to reduce arithmetic errors. OpenAI's GPT-4 system, with its Code Interpreter mode, is a real-world example of a tool-augmented LLM: it can solve challenging math problems by writing and executing code internally. Recent work by Zhou et al. demonstrates that GPT-4 with code execution and self-verification can solve formerly tricky GSM8K problems with very high accuracy [6]. In their approach, the model generates a Python script to compute the result and then double-checks the output, effectively using code as both a calculator and a verifier. Integrating external tools brings a new level of precision to LLM reasoning, addressing the well-known weakness of neural models in exact computation. As these examples show, giving LLMs access to external tools – whether databases or code execution – is a powerful strategy to improve mathematical problem solving. It combines the model's reasoning and language-understanding strengths with the precision and reliability of external systems.

Verifier Models and Answer Re-Ranking: Another strategy, pioneered by Cobbe et al. [7], focuses on improving answer correctness by training separate verifier models to check the solutions generated by LLMs. In this two-stage approach, a large language model (e.g. GPT-3) first generates one or multiple solution attempts for a problem (with or without showing its reasoning), and then a smaller verifier network is trained to classify each candidate solution as correct or incorrect. At test time, the verifier is used to rank the candidates and select the solution most likely to be correct as the model's final answer. This generate-and-verify setup helps catch mistakes that the generative model might make. Cobbe et al. demonstrated a dramatic improvement on GSM8K using verification: their solver with a learned verifier reached 55% accuracy, roughly double that of a baseline GPT-3 model fine-tuned directly on the dataset [7]. The verifier essentially learned to recognize common errors in reasoning or calculation and to prefer solutions that are consistent and likely correct. Notably, verification can compensate for model size to a great extent. An OpenAI study found that a 6B-parameter model with a trained verifier re-ranking its solutions actually outperformed a 175B-parameter model without verification. This corresponds to an effective 30× increase in model size equivalence achieved by using a verifier. Variants of this idea include using the model itself as a verifier – for instance, prompting the LLM to "reflect and check" its answer after producing a solution, or employing a second LLM to critique the first model's answer. In all cases, the principle is not to trust a single pass. Instead, multiple attempts or viewpoints are generated and cross-checked, either by learned models or simple consistency checks, to reduce the incidence of final errors. Today, many state-of-the-art math solvers combine chain-of-thought prompting with a verification stage: they generate numerous solution chains and then either take a majority vote (the self-consistency method from §2.2) or use a verifier network to pick the best chain. Both approaches have a similar effect of boosting reliability through redundancy and evaluation.

Structured and Decomposed Prompting: Researchers have also explored more structured prompting techniques that guide LLMs to solve problems in a modular or staged fashion. One example is decomposed prompting, where a complex problem is broken into a sequence of simpler sub-problems that the model tackles one by one. Rather than posing the full question at once, the user (or an automated system) can prompt the model with a series of sub-queries. In the context of math, Least-to-Most prompting is a related strategy: the model is prompted to first solve some easy, foundational sub-problems, and then use those results to incrementally build up to the final answer [8]. By explicitly ordering the reasoning from simpler to harder, the model is less likely to get

overwhelmed or make leaps of logic. Initial experiments found that least-to-most prompting improved performance on certain math word problems by ensuring that prerequisites are solved before the final answer. Another advanced prompting idea is Plan-and-Solve: here, the LLM is first prompted to outline a solution plan (in natural language or pseudo-code) without producing the final answer yet. Once the plan is sketched out, the model is prompted again to execute that plan step by step to arrive at the answer. This two-phase approach separates high-level strategy from low-level execution, which can lead to more coherent solutions. Yet another extension is to allow non-linear reasoning through methods like Tree-of-Thoughts. Instead of following a single linear chain of thought, the model explores a branching tree of possible reasoning steps, backtracking when a path seems unpromising. Early results showed that with a search algorithm guiding the LLM's choices, this approach can solve some reasoning puzzles more effectively by not committing to a bad step prematurely. These structured prompting techniques are still experimental, but they represent promising directions to increase the reliability and depth of LLM reasoning by imposing more organization on the thought process.

3. Discussion

3.1. Current challenges

Despite the advances above, significant challenges remain before LLMs can solve mathematical problems as reliably as expert humans. This paper highlights three key issues:

Faithfulness and Coherence of Reasoning: There is no guarantee that an LLM's generated chain-of-thought truly reflects correct reasoning. A model might produce a plausible-sounding sequence of steps that do not actually justify the final answer, meaning the explanation is not faithful to the model's real internal process. It may arrive at an answer through pattern matching or luck, then output an illusory rationale. Moreover, LLM solutions often contain missing steps or logical gaps – the model might jump to a conclusion without explaining intermediate steps, or include irrelevant/repetitive steps that do not actually progress the solution. In some cases, the chain-of-thought even has incorrect reasoning yet lands on the correct answer by coincidence, which is problematic because it is hard to trust such a solution. Ideally, every step in the reasoning should be correct and contribute to the result for the right reasons, a standard that current models do not consistently meet. Ensuring that each step is logically valid and that the model "believes" its own step-by-step explanation is an open problem. This challenge is closely related to AI interpretability: methods are required to make the model's reasoning more transparent and truthful to what the model is actually doing to get its answer [9, 10].

Generalization and Robustness: LLMs often struggle to generalize their mathematical reasoning to new or differently phrased problems. A model might perform well on a benchmark like GSM8K that has a certain style of questions, but then fail on a slightly tweaked problem or one drawn from a different source or curriculum. This brittleness arises because the model may latch onto superficial patterns in the training data or prompts, rather than truly learning the underlying math principles. For example, a model fine-tuned on a specific dataset might degrade in accuracy when confronted with problems that have higher linguistic variability or a novel format. In essence, true mathematical reasoning ability should be invariant to how a problem is worded – something current LLMs have not achieved. Extrapolating to entirely novel types of problems (outside the training distribution) is even more difficult. Without explicit new examples, models struggle to apply their knowledge creatively to unfamiliar questions. Thus, robust generalization remains a major hurdle: LLMs need to better capture the fundamental concepts and transferable problem-solving skills, rather than over-fitting to the style of problems they have seen before.

Calculation Errors and Numerical Precision: A persistent issue for LLMs in math is their tendency to make arithmetic mistakes or other computational errors. Unlike a calculator, a language model doesn't perform exact calculations internally – it generates sequences of tokens based on learned probabilities, which means it can output a wrong number if that token happens to be likely in context.

As a result, even a logically correct reasoning approach can be derailed by a minor addition or multiplication error. Studies of models like Minerva found that a large fraction of their remaining errors were due to incorrect arithmetic or algebraic slips. Larger models have improved at basic arithmetic (e.g. a 540B model can do 2–3 digit addition fairly reliably), but they can still falter on long or tricky computations. Without external tools, LLMs cannot guarantee numeric correctness on complex calculations, which is crucial in math domains. Even when an LLM follows a chain-of-thought, it may propagate an early arithmetic mistake through all subsequent steps, leading to an incorrect final answer. Addressing this will likely require hybrid neuro-symbolic systems or specialized training to improve the model's numeracy. In the meantime, incorporating calculators or code execution (as discussed in Section 2.3) is a practical way to mitigate computation errors in LLMs.

3.2. Future prospects

To overcome the above challenges and further improve LLMs' mathematical capabilities, researchers are exploring several promising directions. This paper discusses three complementary approaches, each targeting one of the key issues from Section 3.1:

Structured Reasoning Templates: One idea is to impose more formal structure on the model's reasoning process to ensure coherence and completeness. Instead of having the LLM free-form its entire chain-of-thought in plain language, future systems might employ templates or schemas for solutions – for example, a structured template for a geometry proof or an equation-solving procedure with labeled steps (Given, To Find, Solution Steps, etc.). By constraining the format of reasoning, the model could be guided to produce logically complete and non-redundant solutions, addressing the faithfulness/coherence challenge. Structured templates act like scaffolding to ensure no critical step is omitted or glossed over. Early work on least-to-most prompting and plan-and-solve strategies (discussed above) hints at the benefits of breaking problems into sub-parts in a controlled way. This could be taken further by integrating domain-specific solution frameworks directly into prompts or by designing future LLM architectures that explicitly modularize the reasoning process. For instance, one could imagine separate neural modules for hypothesis generation, calculation, and verification that pass information to each other in a predefined manner. Such neuro-symbolic or modular designs would enforce a higher degree of rigor and transparency in the reasoning pipeline, making the model's solutions more trustworthy and easier to verify step-by-step.

Automatic Verification and Self-Correction: Building on the idea of verifier models, a crucial direction is to make LLMs self-verifying. In other words, after a model proposes a solution, it should internally check each step and the final answer for correctness, much like a mathematician reviewing their own proof. People already see initial steps in this direction: some approaches have the LLM re-evaluate its final answer by plugging it back into the original problem (e.g. verifying a solution by substitution in an equation), while others have the model generate a critique of its own answer and attempt to fix any flaws. Future LLM systems will likely integrate tool-assisted verification even more deeply. For example, an LLM might internally call a math software library or a symbolic theorem prover to confirm a derived result, or use a web search to double-check a factual claim in a word problem. If any check fails, the model could adjust its chain-of-thought and try again, implementing a feedback loop until consistency is achieved. This kind of reflexive reasoning – sometimes called "chain-of-thought with reflection" or self-critiquing – has been shown in preliminary studies to reduce errors. As this area matures, one can imagine an LLM that not only produces a solution but also outputs a confidence level or a formal proof certificate after internally verifying the steps. Automated verification mechanisms, especially when combined with external computational tools for arithmetic or algebra, will greatly increase trust in the model's answers. The end goal is for the model to catch and correct its mistakes before a human has to, resulting in highly reliable outputs. This directly addresses calculation accuracy issues and would significantly boost the reliability of LLMs in mathematical applications.

Enhanced Training and Model Architectures: On a more fundamental level, future research will explore training regimes and architectural changes aimed at overcoming current reasoning limitations.

To tackle the generalization problem, new training objectives and datasets could push LLMs beyond pattern imitation toward more robust logical reasoning. For example, large-scale datasets of formal proofs or synthetic math problems might be created to force models to practice rigorous problem-solving instead of memorizing solution templates. Reinforcement learning from human feedback (RLHF) could be employed with reward signals that emphasize not just getting the correct answer but doing so via a correct reasoning process (penalizing contradictions or gaps in the chain-of-thought). In terms of architecture, incorporating symbolic or memory-augmented components is a promising avenue. Neural architectures that integrate with symbolic theorem provers, or that maintain an explicit memory of intermediate results and known facts, could allow a model to reason over these stored facts and reuse intermediate computations, rather than treating every problem as a blank slate. There is already work on memory-augmented Transformers and modular neural networks for reasoning tasks. Applying these ideas to mathematical reasoning—designing models that inherently respect logical consistency and have a form of working scratchpad for calculations—could substantially improve both generalization and reliability. In short, by training models with more targeted data and objectives, and by building in architectural features that support stepwise reasoning, the current studies hope to create next-generation LLMs that generalize better to new problems and maintain consistency in their reasoning.

4. Conclusion

In conclusion, large language models have begun to demonstrate impressive – if still limited – capabilities in mathematical problem solving. Techniques like chain-of-thought prompting have unlocked latent reasoning abilities in LLMs, enabling them to tackle problems that were once far out of reach for purely language-based models. Further innovations such as self-consistency decoding, retrieval augmentation, and tool integration have pushed performance to new heights on challenging math benchmarks, in some cases rivaling or surpassing earlier specialized models. Through this survey, it can be observed that each strategy addresses particular weaknesses of LLMs (be it the inability to break down complex tasks, the tendency to hallucinate facts or make arithmetic mistakes, etc.), incrementally improving the model's overall competence in mathematics.

However, this review also makes clear that significant gaps remain between LLMs and human mathematicians. Issues of reasoning transparency, reliability, and generalization continue to pose obstacles. The current generation of models can still be tripped up by complex multi-step problems or can produce solutions that are correct by coincidence rather than by solid logic. As such, improving the mathematical reasoning capabilities of LLMs is both of high practical value and of deep research significance. Progress in this area will not only lead to better performance on math tasks, but also inform about how to instill more general problem-solving and reasoning skills in AI systems. Advances in mathematical reasoning could translate into more trustworthy AI assistants in many domains requiring rigorous thought – from scientific research support to advanced tutoring systems.

Finally, efforts to enhance LLMs in mathematics carry broader implications for AI safety and interpretability. A model that can explain its reasoning step by step – and have those steps verified – is inherently more transparent and easier to trust. Thus, work on mathematical reasoning aligns with the goal of making AI systems whose decision-making processes are understandable and verifiable by humans. In summary, strengthening the math problem-solving abilities of LLMs is a crucial step toward AI that is not only more capable, but also more aligned with human expectations of correctness and clarity. Ongoing research combining algorithmic innovations with collaborative and structured reasoning frameworks gives reason to be optimistic that future LLMs will overcome many of today's limitations, ushering in an era where AI can reliably partner with humans in solving even the most complex quantitative problems.

References

[1] J. Wei, et al., Chain-of-thought prompting elicits reasoning in large language models. Adv. Neural Inf. Process. Syst. 35, 24824–24837 (2022).

[2] T. Kojima, et al., Large language models are zero-shot reasoners. Adv. Neural Inf. Process. Syst. 35, 22199–22213 (2022).

[3] X. Wang, et al., Self-consistency improves chain of thought reasoning in language models. Proc. Int. Conf. Learn. Represent. (ICLR 2023).

[4] Z. Wang, et al., RAT: Retrieval-augmented thoughts elicit context-aware reasoning in long-horizon generation. arXiv: 2403.05313 (2024).

[5] L. Gao, et al., PAL: Program-aided language models. Proc. 40th Int. Conf. Mach. Learn. (ICML 2023).

[6] A. Zhou, et al., Solving challenging math word problems using GPT-4 code interpreter with code-based self-verification. arXiv: 2308.07921 (2023).

[7] K. Cobbe, et al., Training verifiers to solve math word problems. arXiv: 2110.14168 (2021) (presented at ICLR 2022).

[8] D. Zhou, et al., Least-to-most prompting enables complex reasoning in large language models. arXiv: 2205.10625 (2022).

[9] P. Linardatos, V. Papastefanopoulos, S. Kotsiantis, Explainable AI: A review of machine learning interpretability methods. Entropy 23, 18 (2020).

[10] V. Vishwarupe, P.M. Joshi, N. Mathias, S. Maheshwari, S. Mhaisalkar, V. Pawar, Explainable AI and interpretable machine learning: A case study in perspective. Procedia Comput. Sci. 204, 869–876 (2022).