nature machine intelligence

Perspective

https://doi.org/10.1038/s42256-025-01092-w

# Quantifying artificial intelligence through algorithmic generalization

Received: 3 October 2024

Accepted: 18 June 2025

Published online: 18 August 2025

Check for updates

Takuya Ito ✉, Murray Campbell, Lior Horesh, Tim Klinger & Parikshit Ram

The rapid development of artificial intelligence (AI) systems has created an urgent need for their scientific quantification. While their fluency across a variety of domains is impressive, AI systems fall short on tests requiring algorithmic reasoning—a glaring limitation, given the necessity for interpretable and reliable technology. Despite a surge in reasoning benchmarks emerging from the academic community, no theoretical framework exists to quantify algorithmic reasoning in AI systems. Here we adopt a framework from computational complexity theory to quantify algorithmic generalization using algebraic expressions: algebraic circuit complexity. Algebraic circuit complexity theory—the study of algebraic expressions as circuit models—is a natural framework for studying the complexity of algorithmic computation. Algebraic circuit complexity enables the study of generalization by defining benchmarks in terms of the computational requirements for solving a problem. Moreover, algebraic circuits are generic mathematical objects; an arbitrarily large number of samples can be generated for a specified circuit, making it an ideal experimental sandbox for the data-hungry models that are used today. In this Perspective, we adopt tools from algebraic circuit complexity, apply them to formalize a science of algorithmic generalization, and address key challenges for its successful application to AI science.

The recent evolution of modern artificial intelligence (AI) systems and large language models (LLMs) has led to the speculation that these systems may reason[1–4]. However, owing to challenge of evaluating large models trained on massive pretraining datasets[5], it is difficult to evaluate whether such models are truly exhibiting algorithmic reasoning abilities, or whether they instead regurgitate plausible text from their pretraining data[6,7]. This ambiguity has led to a deluge of reasoning benchmarks[8–17]. Despite these efforts, objectively quantifying the complexity of reasoning problems is difficult; most of these experiments are ad hoc, and designed without a framework to quantify and verify the algorithmic complexity of reasoning problems. However, approaches in computational complexity theory, a field within theoretical computer science, have made it possible to explicitly measure a problem's algorithmic difficulty, paving the way for generalization tests rooted in quantifiable measures of complexity. In this Perspective, we bridge algorithmic reasoning with a decades-old branch of computational complexity—circuit complexity theory—to provide a theoretical link to studying the complexity of algorithmic computation in modern AI systems.

Recently, there has been increased interest in studying modern AI models through arithmetic and compositional tasks[9,12,18–27]. Compositional tasks are problems rooted in a long history from the early twentieth century[28,29] that are generated by recombining a basis set of atomic elements to form a variety of task combinations (for a review, see Russin et al.[30]). (Arithmetic problems are compositional; they are composed of atomic elements (numbers and operators), and can be recomposed to generate novel expressions and problems.) For modern AI systems, compositional tasks can serve as useful reasoning

Mathematics and Theoretical Computer Science Department, T.J. Watson Research Center, IBM Research, Yorktown Heights, NY, USA.
✉e-mail: taku.ito1@gmail.com

https://doi.org/10.1038/s42256-025-01092-w

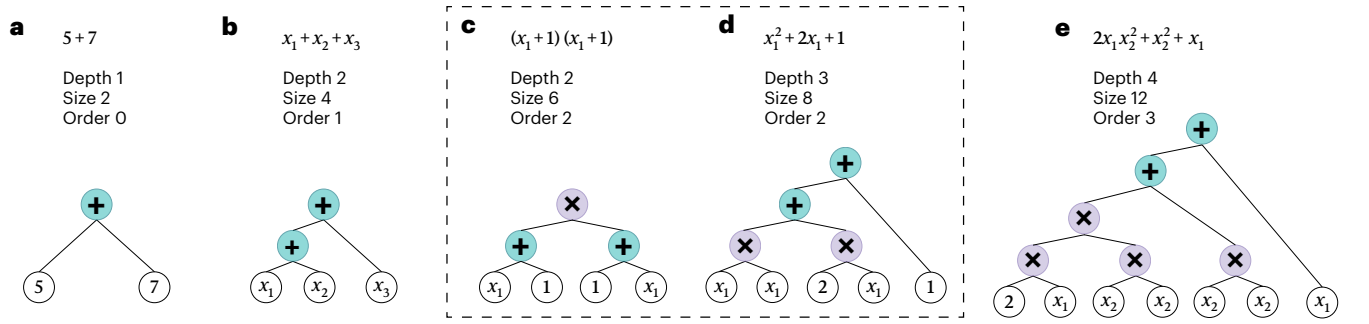


**Fig. 1 | Examples of algebraic expressions represented as circuits.** **a**, A two-operand addition circuit (input gates are sampled from a field $\mathbb{F}$). **b**, A three-operand addition circuit (input gates are sampled from the set of variables $x_i \in X$ rather than $\mathbb{F}$). **c**,**d**, A mathematically equivalent pair of circuits, but represented as a factorized expression (**c**) and its a monomial expansion (**d**). Notably, despite their mathematical equivalence, the circuit representations are distinct. **e**, A polynomial of depth 4 and size 12.

benchmarks as they require (1) abstraction, (2) logical and verifiable application of rules or axioms, and (3) precise problem-solving and rigour. Critically, these paradigms have provided reliable ways to elicit failure modes in transformer-based AI models for specific forms of compositional generalization. For example, a number of studies have demonstrated the difficulty of 'length generalization'—generalizing to problems of longer length than seen during training[19,21,26,31]. Other researchers have also introduced various notions (for example, systematicity and productivity) in an effort to taxonomize different forms of compositionality[9,32–35]. By contrast, the formalisms from circuit complexity theory provide a set of tools that can be applied to quantify algorithmic generalization—generalization over algorithms specified by circuits and their associated measures of complexity, such as space or time complexity. (Structural properties of circuits correspond directly to algorithmic requirements to compute that problem.) Moreover, formalizing problems through a circuit complexity framework provides a theoretically grounded framework for the increasingly popular yet nascent empirical evaluations in AI systems that use arithmetic and compositional tasks[18,19,21–24,36–41]. Although we focus on formalizing algebraic problems through the lens of circuit complexity (that is, algebraic circuit complexity) due to the widespread use of arithmetic problems to evaluate modern AI systems, the more general framework of circuits can be similarly extended to other algorithmic problems.

A large class of algorithmic problems can be studied with algebraic expressions[42,43]. Algebraic circuit complexity theory formalizes the evaluation of algebraic expressions through algorithms encoded as computable circuits (that is, directed acyclic graphs; Fig. 1). This formalization is well established in computational complexity theory, the branch of theoretical computer science concerned with quantifying the difficulty of computational algorithms and the resources required to solve them[44]. Importantly, formalizing computational problems in terms of circuits is the leading approach to empirically quantify their complexity. Unlike other notions of complexity, such as Kolmogorov complexity in algorithmic information theory (which is incomputable), notions developed in complexity theory for circuits are explicitly computable and determined by their shape and structure[45]. Thus, the tools of circuit complexity can formalize notions of generalization over algorithms by defining benchmarks in terms of their circuit properties. Furthermore, algebraic circuits are generic mathematical objects; they can be represented from a variety of mathematical perspectives (geometry, topology and so on), providing useful interpretations in other domains. Algebraic circuits are therefore well situated to investigate algorithmic generalization—the problems are computable and verifiable, large datasets can be straightforwardly generated from circuit specifications, and new models can be developed that address specific failure modes within this framework. In the ensuing sections, we provide a blueprint for the successful adoption of algebraic circuit complexity as a gateway into studying algorithmic generalization more broadly. We introduce the core components of algebraic circuits, address how they can be leveraged to study algorithmic generalization, and discuss several key open theoretical and empirical challenges.

## Algebraic circuits

Algebraic circuit complexity studies algebraic expressions as computable circuit models. There has been a substantial amount of recent machine learning research studying arithmetic generalization, a key algorithmic ability. While important and insightful, most of those studies primarily focus on a restricted set of algebraic problems, which we will illustrate later[18,19,21,24,39]. Below, we provide definitions of algebraic circuits that will place previous work within a broader mathematical framework. Our goal is to provide the tools to quantify model generalization through circuit complexity, rather than ad hoc goals such as length generalization.

### Definition

(For a more formal definition, see Shpilka and Yehudayoff[42] or Bürgisser et al.[43].) An algebraic circuit $C$ represents a polynomial expression as a directed acyclic graph, composed of gates $v$ and edges $e$. Input gates, $v_X$ and $v_{\mathbb{F}}$, are defined as either variables (for example, $X = \{x_1, \ldots, x_n\}$) or elements in a field $\mathbb{F}$ (for example, $\mathbb{R}$), respectively. Input gates have a fan-in (in-degree) of 0. All other gates are operators: a sum gate ($v_+$) or a product gate ($v_\times$). In this Perspective, we restrict the fan-in of operator gates to 2, as is standard[42]. For our purposes, this ensures that there is the same number of gates in a circuit model with its corresponding representation as a string of tokens, which simplifies downstream analyses (for example, the analysis in Fig. 6). Operator gates have a fan-out (out-degree) of either 1 or 0. If the fan-out of an operator gate is 0, then it is the output gate of that polynomial.

### Properties

An algebraic circuit $C$ has two main algorithmic complexity measures: size $s$ and depth $d$ (see Table 1 for a summary of all properties). The size $s$ of a circuit refers to the number of edges $e$ in $C$, and corresponds to algorithmic space complexity. The depth $d$ of a circuit refers to the longest path from an input gate to the output gate, and corresponds to algorithmic time complexity. We can denote a subcircuit $C_v$ of $C$, which computes the polynomial $f_v$ rooted at gate $v$. Another important property of an algebraic circuit is its degree (that is, the degree of a polynomial). The degree of a circuit (or a subcircuit) can be computed by measuring the degree of the gate $v$, denoted $\deg(v)$. Elements in a field $\mathbb{F}$ are of degree 0, input variables $x \in X$ have degree 1, the degree of an sum gate (+) is determined by the degrees of its inputs $u$, $v$ such that $\deg(u + v) = \max(\deg(u), \deg(v))$, and the degree of a product gate ($\times$) is determined by $\deg(u \times v) = \deg(u) + \deg(v)$.

**Table 1 | Key circuit properties and their descriptions**

| Circuit property | Description |
|---|---|
| Circuit $C$ | A directed acyclic graph that computes a polynomial. |
| Size $s$ | The number of edges in $C$. |
| Depth $d$ | The longest path from an input gate to an output gate. |
| deg($v$) | The degree of a gate $v$. If $v$ is the output gate, deg($v$) is the degree of the polynomial. |
| Gate $v_{\mathbb{F}}$ | An element of a field $\mathbb{F}$ with $\deg(v_{\mathbb{F}}) = 0$, and fan-in 0. |
| Gate $v_X$ | A variable $x_i \in X$ with $\deg(v_X)=1$, and fan-in 0. |
| Gate $v_+$ | A sum gate with $\deg(v_+)=\max(\deg(u),\deg(v))$ for inputs $u$ and $v$, and fan-in 2. |
| Gate $v_\times$ | A product gate with $\deg(v_\times)=\deg(u)+\deg(v)$ for inputs $u$ and $v$, and fan-in 2. |

Figure 1 provides a few simple examples of algebraic circuits. A circuit representation of an algebraic expression provides a concrete algorithm for computing a polynomial, with some circuits requiring more computation than others (for example, determined by size or depth). Furthermore, a circuit description can provide explicit differences in required computation for polynomials that are mathematically equivalent (for example, Fig. 1c,d). Such a formalization can be useful to study how different representations of equivalent algebraic expressions can lead to different levels of generalization, or how syntax relates to semantics in formal languages. The recent studies in arithmetic generalization in AI models focus on the simplest circuit representations of algebraic problems (problems analogous to Fig. 1a, but varying the magnitude of the field elements, for example, train on small digit numbers, test on large digit numbers). The language of circuit complexity can provide more interesting and flexible ways to investigate generalization across circuit complexity classes.

## Towards a science of generalization with algebraic circuits

A recent study concluded that the behaviour of LLMs is a function of the problems they are trained to solve[25]. It is difficult to identify what the 'problems' are when the pretraining corpus is natural language text derived from the internet. A theoretically coherent alternative to quantify AI systems is to select problems for which (1) complexity is quantifiable and (2) arbitrarily large datasets can be systematically generated. Algebraic circuits satisfy both of these constraints.

Although recent papers have demonstrated an increased interest in using arithmetic tasks to quantify generalization, preliminary approaches have been theoretically limited. As alluded to above, many of the recent papers evaluating length generalization in arithmetic tasks train on two-variable addition (or multiplication), and test on the same circuit class but sample field elements that are larger in magnitude[19–21,23,26,39,41] (Fig. 2a). Others focus on a more complex form of generalization, that is, addition or multiplication on problems with a greater number of variables, which is equivalent to increasing the circuit size and depth (Fig. 2b; for example, a modular arithmetic task[36]). While useful and informative, both these tests of generalization scratch the surface of generalization metrics that can be devised with algebraic circuits.

### The importance of learning composable functions

We first define algorithmic generalization (Box 1), and consider the importance of learning algorithmically. The ability to generalize algorithmically is one of the most challenging problems in AI, and is a requirement for robust reasoning and planning. Like algebraic circuits, several previous papers formulated algorithmic tasks as computing a path through a directed graph or circuit[11,13,24,46]. Importantly, learning an algorithmic task requires (1) decomposing and abstracting the individual functions (for example, gates) of a circuit, and (2) understanding how they can be recomposed via edges to produce novel circuits (for example, function composition). Algebra is a natural language to study algorithmic generalization as it can be encoded as a circuit, and requires both abstraction and function composition. Moreover, in contrast to other compositional approaches (for example, regular or context-free grammars[36,47]), algebraic problems encompass an infinite vocabulary (for example, $\mathbb{R}$) with an expressive grammar determined by its axioms, making it well suited for machine learning algorithms that require many training samples. In the following section, we propose an algorithmic generalization framework in terms of algebraic circuits.

### Circuit divergence as a metric of generalization

The language of algebraic circuits provides quantitative metrics to formalize generalization in terms of algorithmic complexity, as circuit structure directly corresponds to algorithmic properties. In particular, generalization of AI systems can be quantified in terms of circuit divergence—the divergence of circuit parameters between algebraic circuits employed during training and testing (Fig. 3). This emphasis on characterizing the algorithmic properties of problems complements previous work focused on establishing metrics of compositionality based on the degree of subgraph overlap in training and testing sets. One particularly relevant metric is maximum compound divergence (MCD) introduced by Keysers et al.[17], which measures the divergence of the frequency distribution of compositional subgraphs between the training and testing sets. While Keysers et al.[17] demonstrated the empirical relevance of MCD on generalization performance, here we emphasize the importance of designing training and testing partitions according to the distribution of algorithmic complexity (that is, circuit) properties. This contrasts with MCD, which focused on providing a single summary statistic that, although empirically useful, does not distinguish between algorithmic properties. Specifically, designing benchmarks by manipulating circuit size (versus depth) enables the characterization of generalization over algorithmic size (versus time) complexity. This in turn can help practitioners isolate how specific architectural components of models map onto algorithmic capabilities: for example, circuit depth of a problem typically relates to the number of layers of a transformer, while circuit size relates to the degree of parallelization, such as context length size for transformers[48].

Given a model $M$, we seek to characterize its generalization performance $M(C_{\text{test}}|\mathcal{C})$, where $C_{\text{test}}$ is a test circuit, and $\mathcal{C} = \{C_1, \ldots, C_k\}$ is a family of training circuits. We define circuit divergence as the difference between quantifiable circuit properties between train and test distributions. Here we emphasize five important circuit properties to measure divergences: size $s$, depth $d$, the polynomial degree, the sampling of a field $\mathbb{F}$, and the number of variables $|X|$. By quantifying the properties of circuits in both the training and testing sets, circuit divergence can be explicitly measured along these dimensions. In the following sections, we provide several examples for which tests of generalization can be constructed through the manipulation of circuit divergence.

### Generalization benchmarks

In this section, we illustrate the flexibility of algebraic circuits in designing meaningful AI benchmarks. To demonstrate the generality of algebraic circuits, we begin by providing benchmarks that are analogous to popular tests of compositional generalization: systematicity and productivity[9]. We subsequently introduce novel, more challenging problems that can be used to evaluate more general types of algorithmic computation and abstraction. Finally, we demonstrate an approach to reformulate tests of algorithmic generalization for pretrained LLMs.

**Compositional generalization with algebraic circuits.** There has been recent interest in using compositional paradigms to systematically study the generalization capabilities of machine learning

https://doi.org/10.1038/s42256-025-01092-w

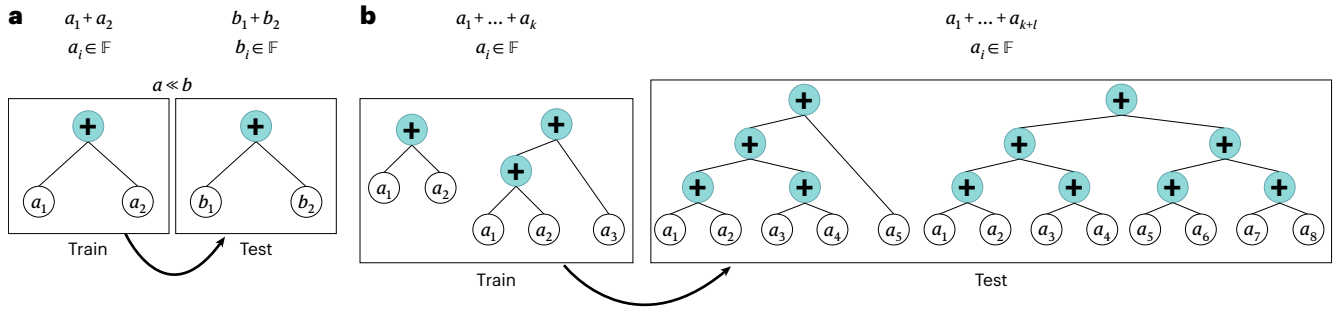


**Fig. 2 | Commonly used AI evaluations for length generalization with arithmetic tasks. a**,The predominant form of 'length generalization' in transformers is evaluating performance on addition or multiplication problems with larger integers than seen during training[19–21,23,26,39]. This would be conceptually equivalent to a situation in which the circuit size and depth are fixed, but the sampling of input gates (that is, field elements) differs across training and testing sets. (However, see discussion in 'Open theoretical and empirical challenges' on the nuances of computing long addition.) The notion of length generalization is specific to the context of transformers, given that larger digits require a larger context window. **b**, Another form of length generalization studied in the literature is to generalize to arithmetic problems with more operands (variables) than seen in the training set[36]. From a circuit complexity perspective, these two approaches are distinct problems.

models[8,9,12,24,46,49–54]. Here we demonstrate direct links to common forms of compositional generalization using algebraic circuits.

*Systematic compositional generalization and regression*. Systematic compositional generalization refers to the ability to recombine known basis elements into novel combinations of fixed sequence size (Fig. 4a). This means that test sets of systematicity are limited to novel combinations of the same length as seen during training[8,9]. The analogue in algebraic circuits is (1) to sample a family of circuits $\mathcal{C}$ over a field $\mathbb{F}$ of a fixed size and depth, and (2) to generalize to circuits of the same size and depth, but differentially sampling input gates and/or operators. This can be implemented by choosing different samplers—$P_1$ and $P_2$—that differentially sample gates (Fig. 4a). For example, the training set of circuits could be constructed with input gates $a_i \in_{P_1} \mathbb{F}$, and the testing set of circuits could be constructed with input gates $b_i \in_{P_2} \mathbb{F}$. (Note that when samplers preferentially choose $b_i \gg a_i$, this is analogous to the common test of length generalization; Fig. 2a.)

In real-world data, the distributional properties of the training distribution can often bias AI (or even human) learners towards learning memorized short cuts, rather than learning algorithmic or syntactic strategies that enable robust generalization[7]. However, given its infinite vocabulary and the ability to sample circuits with well-defined algorithmic properties, algebraic circuit complexity provides a comprehensive experimental sandbox for designing and sampling from diverse data distributions. Through this sandbox, practitioners can identify specific properties of train–test distributions that can either introduce a distributional bias or confound to be assessed, or ameliorate these biases by counterbalancing the distribution. Interestingly, we note that successful systematic compositional generalization on input gates sampled from different distributions amounts to learning a distributionally robust regression model (for example, Zhang et al.[55] and Ghosh et al.[56]). This is because a circuit of specific size and depth (for example, as shown in Fig. 4a) expresses a specific polynomial. Generalization over a fixed circuit with a distribution shift of its input gates (for example, field elements but not operator gates) would demonstrate successful out-of-distribution generalization.

Previous notions of systematic compositional generalization have specified 'weak' and 'strong' forms of systematicity[57]. Strong systematicity refers to the ability to generalize to tokens (for example, operands) in novel syntactic positions. (For example, if a model is exclusively trained on expressions of the form $a + b$ and $a + c$ where $a$ is always the first token, a model that generalizes to $b + a$ where $a$ is the last token would exhibit strong systematic generalization.) Weak systematicity refers to the ability to exclusively generalize to novel expressions in which $a$ is always in the same syntactic position (for example, always the first token). While this distinction does not alter the semantics for commutative algebras (as discussed in this paper), this distinction is important for many formal languages in which permuting syntactic ordering produces novel semantics, such as in many context-free grammars and non-commutative algebras. (In these cases, circuits can be encoded as ordered directed acyclic graphs, in which the order of parent/input gates is preserved.)

*Productive compositional generalization*. Productive compositional generalization refers to the ability to generalize to sequences of longer length. In the context of algebraic circuits, while systematicity focuses on keeping circuit size and depth fixed while manipulating gates, productive generalization focuses on manipulating circuit size and depth (Fig. 4b). Thus, evaluating systematic and productive generalization can be studied together; measures of generalization can be quantified in a continuous manner by varying circuit parameters, such as gate samplers (systematicity), size (productivity) and depth (productivity). Understanding how each of these properties interact across training and testing sets will provide a comprehensive quantification of generalization over distinct algorithmic properties.

As algebraic circuits are circuit representations of algebraic expressions, one can ask more generic questions about algebraic polynomials. For example: given a class of polynomials as a training dataset, what other class of polynomials will this model be able to compute? Such a question goes beyond asking whether a model can systematically or productively generalize. Instead, it addresses a basic question that can leverage other rich mathematical subfields (for example, geometry, topology) to quantify algorithmic generalization. Algebraic circuits provide a flexible framework to formalize problem complexity beyond existing paradigms in compositional generalization.

**Classification tasks: polynomial identity testing.** Previous work studying arithmetic abilities in transformer models has typically focused on computing simple expressions with field elements ($a \in \mathbb{F}$; for example, $5 + 7 = ?$), rather than abstract variables ($x \in X$; for example, $2x_1 + x_2 + 7$). Including variables in an algebraic expression increases the polynomial degree ($d \geq 1$), thereby increasing its complexity and the need for abstractions. One approach to evaluating algebraic circuits with variables is through polynomial identity testing, an active area of research in computational complexity theory and computational algebra. Polynomial identity testing evaluates whether two polynomials are equivalent (that is, $P_1(x_1, \ldots, x_n) \equiv P_2(x_1, \ldots, x_n)$). Posed another way, one can ask whether $P_1(x_1, \ldots, x_n) - P_2(x_1, \ldots, x_n) \equiv 0$.

This problem can be naturally formulated as a binary classification task for AI models. Importantly, studying the extent of algebraic

## BOX 1

# A formal sketch of algorithmic generalization

Let $B$ be a basis set of elements. Let $U_1 = B$ and $U_{i+1} = \{u | (u_1, u_2 \in U_{\leq i})$ and $(u = u_1 \circ u_2)\}$, where $\circ$ stands for any computable composition operator on $U_i$ (akin to production rules in context-free grammars or operations in a field). Then, we define some universe set

$$U = \bigcup_{i \in \mathbb{N}} U_i$$

We define a function (task) $T$ on $U$ such that $T: U \to R$ (see below for tasks $T$ for algebraic circuits). $T$ is a compositional function, where $\forall u_1, u_2 \in U, T(u_1 \circ u_2) = T(u_1) * T(u_2)$, where * stands for any computable composition on $R$.

Let $D_{\text{train}}$ and $D_{\text{test}}$ be distributions over $U$, and $\text{supp}(D_{\text{train}}|B)$ and $\text{supp}(D_{\text{test}}|B)$ to be the support of the basis set elements in $D_{\text{train}}$ and $D_{\text{test}}$, respectively. We restrict $D_{\text{test}}$ to be a distribution in which $\text{supp}(D_{\text{test}}|B) \subseteq \text{supp}(D_{\text{train}}|B)$. In addition, if a composable function $\circ$ is represented in $D_{\text{test}}$ (that is, $u \in D_{\text{test}}$ and $u = b_1 \circ b_2$), then we require that $\circ$ is also represented in $D_{\text{train}}$. We include these as requirements for algorithmic generalization; it would be challenging, if not impossible, to generalize to samples in $D_{\text{test}}$ if not all basis elements and composition operators were provided in $D_{\text{train}}$.

We define algorithmic generalization as the evaluation of a learned model $M$ (for example, a neural network) where

$$\Pr_{x \sim D_{\text{test}}}[M(x) = T(x)] > 1 - \epsilon \quad (1)$$

Here, $M$ is only optimized from samples $x \sim D_{\text{train}}$, and $\epsilon$ is the evaluation error of $M$ on $x \sim D_{\text{test}}$. A strong form of algorithmic generalization over algebraic circuits would be to satisfy equation (1) for all valid partitions of $D_{\text{train}}$ and $D_{\text{test}}$ (with the requisite support of basis sets and composition functions) in $U$ for uniformly small values of $\epsilon$.

For algebraic circuit problems, $T: U \to R$ can be a function that maps a circuit to: (1) a field element, for example, $r \in \mathbb{R}$ (in the case of circuit evaluation; Fig. 1); (2) $\{0, 1\}$, in the case of a classification task, such as polynomial identity testing (Fig. 3a); (3) another circuit $C$ (that is, in the case of polynomial expansion or factorization; Fig. 3b).

generalization by assessing train–test circuit divergences of polynomial identity problems can be rigorously quantified, as each identity problem can be encoded as a circuit (Fig. 5a). The complexity of a polynomial identity testing problem can be scaled up by increasing the size, depth and/or degree of the test problems. Moreover, machine learning practitioners can leverage existing symbolic programs and solvers to systematically generate large datasets (for example, SymPy; Meurer et al.[58]). Polynomial identity testing thus provides a unique opportunity to study AI generalization in terms of circuit complexity theory.

**Sequence-to-sequence tasks: polynomial expansion or factorization.** A wide application of generative models is in sequence-to-sequence transduction tasks. One particularly impactful use case is the development of AI models for code. AI models for code take in code as inputs (for example, the programming language COBOL), and generate a translation of that code (for example, Java). Despite its potential importance for modernizing many existing codebases, there is substantial scepticism as to whether generative models trained with next-token prediction can reliably generate accurate code translations. This is due to the fact that programming languages are not dictated by autoregressive processes, and instead governed by algorithmic rules.

Learning on algebraic circuits provides a straightforward framework to evaluate the ability of models to learn sequence-to-sequence tasks that are governed by algorithmic rules. For example, the problem of expanding or factorizing a polynomial is a problem that is governed by the axiomatic rules of algebra (Fig. 5b). Like code translation, this task requires transforming one sequence into another while maintaining mathematical equivalence (or for code, semantic equivalence despite syntactic differences). Importantly, there are explicit tools to describe the complexity with which the translation occurs in algebraic circuits. For example, a polynomial expansion transforms a factorized representation (that is, shallow circuit) to a sum of monomials representation (that is, deeper circuit). In contrast, a polynomial factorization implements the inverse operation, which requires compressing a large circuit to a smaller circuit. While a general interesting question is to ask whether models trained on one type of transformation can learn the other, this formalization has natural implications for understanding how to design AI models for code. In particular, some programming languages may have lower-level syntax (for example, COBOL) relative to other languages, such as Python or Java. In either case, understanding how lower-level and higher-level encodings of an expression (be they algebraic or programmatic) requires learning useful abstractions and the algorithms to translate them. Thus, studying the conditions by which AI models can robustly parse and translate algebraic circuits can shed light on the best strategies to train AI models to translate programming languages.

### Mechanistic interpretability with algebraic circuits

A major issue in assessing algorithmic generalization is the difficulty of interpreting what goes awry when they fail to generalize. This is partly due to the lack of interpretability of many benchmarks, which are often presented in natural language, and for which verifiable algorithms (for example, circuit diagrams or parse trees) do not exist[14,59]. A more recent set of approaches in characterizing the interpretability of neural network representations rely on the design of carefully constructed tasks that have interpretable tasks, such as ground-truth parse trees or circuit diagrams[24,36,60]. Algebraic circuits similarly provide verifiable tasks and subtasks. While the input to an AI model might be a string representation of a polynomial, that polynomial's circuit encoding provides its ground-truth encoding. Such an encoding makes it easy to directly compare to the internal representations of a model, such as the attention weights within a transformer (Fig. 6). This makes it possible to track if the internal representations of a transformer computes the algebraic expression similarly to the algorithm encoded by its circuit—the normative algorithm to computing polynomials. More broadly, the distance metric that computes the distance between attention weights and ground-truth circuit edges (as shown in Fig. 6) can be used as a regularization term to encourage transformers to learn normative circuit computations, providing algorithmic and step-by-step information to the model. Overall, evaluating AI models on algebraic problems allows practitioners to adjudicate between theoretical models of circuit computation with modern AI computation.

### Evaluating LLMs with algebraic circuits

We have introduced algebraic circuit tasks within the context of training models from scratch. While it is difficult to evaluate pretrained LLMs with precise levels of certainty due to the obscure nature of the pretraining data and optimization protocols, algebraic tasks can still be repurposed for LLMs. Previous work has demonstrated that LLMs typically fail on mathematical problems, which require abstract reasoning on variables[7,25,61]. Thus, if an LLM calls specific tools (for example, calculators) that make computing circuits with only field elements too simple (for example, those illustrated in Figs. 1 and 2), we can

https://doi.org/10.1038/s42256-025-01092-w

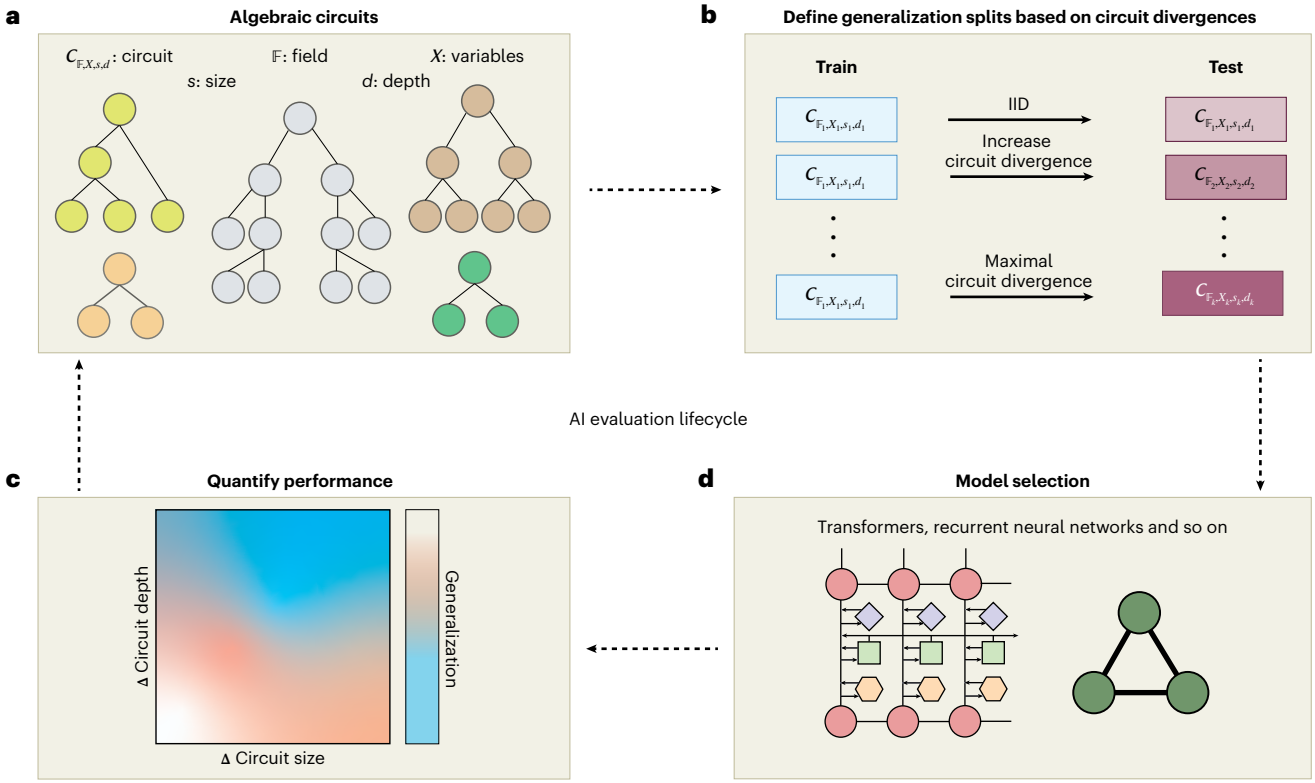


**Fig. 3 | Algorithmic capabilities of modern AI systems and architectures can be studied with algebraic circuits. a**, A set of problems can be identified (and sampled) from algebraic circuits. **b**, Given a family of circuits, we can identify circuit divergences—the divergence of different circuit properties, such as size and depth—to design train and test datasets to evaluate the algorithmic generalization capabilities of a model. IID, independent and identically distributed. **c**,**d**, We can evaluate a model (or set of models) across these circuit splits (**d**) and quantify their performance according to circuit divergences (**c**). This AI evaluation lifecycle allows us to iterate and refine hypotheses regarding the degree to which a model can generalize to a class of algorithms. Note that circuit divergences can be measured beyond depth and size.

replace these with circuits containing variables $x \in X$ (that is, those of degree ≥1). For example, such tasks can include the aforementioned polynomial identity testing or polynomial expansion and factorization tasks, both of which can be made arbitrarily difficult and use arbitrary sets of variables and tokens.

Given that previous studies have demonstrated poor abstract reasoning abilities of LLMs, it is unlikely that current LLMs will be able to compute arbitrary circuits out-of-the-box. However, recent techniques in prompting LLMs have suggested the ability of LLMs to learn from a few prompts (termed in-context learning)[37,62–67]. In other words, a few question–answer pairs can be shown to the LLM in the context window, followed by the target problem. This approach naturally offers an approach to measure algorithmic generalization across circuit divergence metrics: given a class of question–answer pairs generated from a specific circuit class, how well can a model generalize to a problem generated from a different complexity class (Fig. 7)? Another approach to systematically evaluate the ability of LLMs to compute circuits is through chain-of-thought prompting. Recent theoretical studies have indicated that leveraging chain-of-thought enables LLMs to solve more challenging problems by enhancing their expressivity and using scratchpads[48,68–70]. Practically, inducing chain-of-thought in LLMs provides a way to verify whether LLMs sequentially 'reason' through steps that are analogous to ground-truth circuit computations (that is, following the edges from input gates). Moreover, as algebraic circuits encode an explicit algorithm to compute an expression, each step through the circuit produces a verifiable intermediate computation. This enables the evaluation of LLMs not only on verified outputs but also on verified intermediate reasoning chains, providing an opportunity to evaluate reinforcement learning approaches that have been increasingly used to develop reasoning-based models[4,71]. Together, these experimental approaches provide concrete methods from which to quantify algorithmic ability in LLMs.

## Open theoretical and empirical challenges

Circuits provide a useful formalism to study algorithmic generalization in AI systems. They are also a leading approach to quantifying the computational complexity of algorithmic problems in theoretical computer science. Previous AI reasoning literature has focused on benchmarking AI to human-level reasoning[72–75], or on how humans learn algorithmic tasks[49,76,77], both of which are important areas of investigation from the perspectives of cognitive science and AI. However, this Perspective provides a complementary viewpoint from theoretical computer science aimed to quantify reasoning based on generalizing over algorithmic complexity. Here we highlight the primary challenges associated with linking formal circuit models of computation with AI generalization.

### Circuit complexity and AI generalization

Algebraic circuit complexity studies the algorithmic resources required to evaluate polynomials. Although there are other metrics of complexity, such as Kolmogorov complexity in algorithmic information theory[78,79], this measure of complexity is incomputable, as it requires searching over an infinite number of programs (although there are efforts in approximating Kolmogorov complexity with alternative methods; Wyeth and Sturtivant[45], Johnston et al.[80] and Dingle et al.[81]). Thus, while circuits allow for the explicit computation of a problem, it remains unclear as to whether notions from circuit complexity will naturally map onto notions of AI generalization.

https://doi.org/10.1038/s42256-025-01092-w

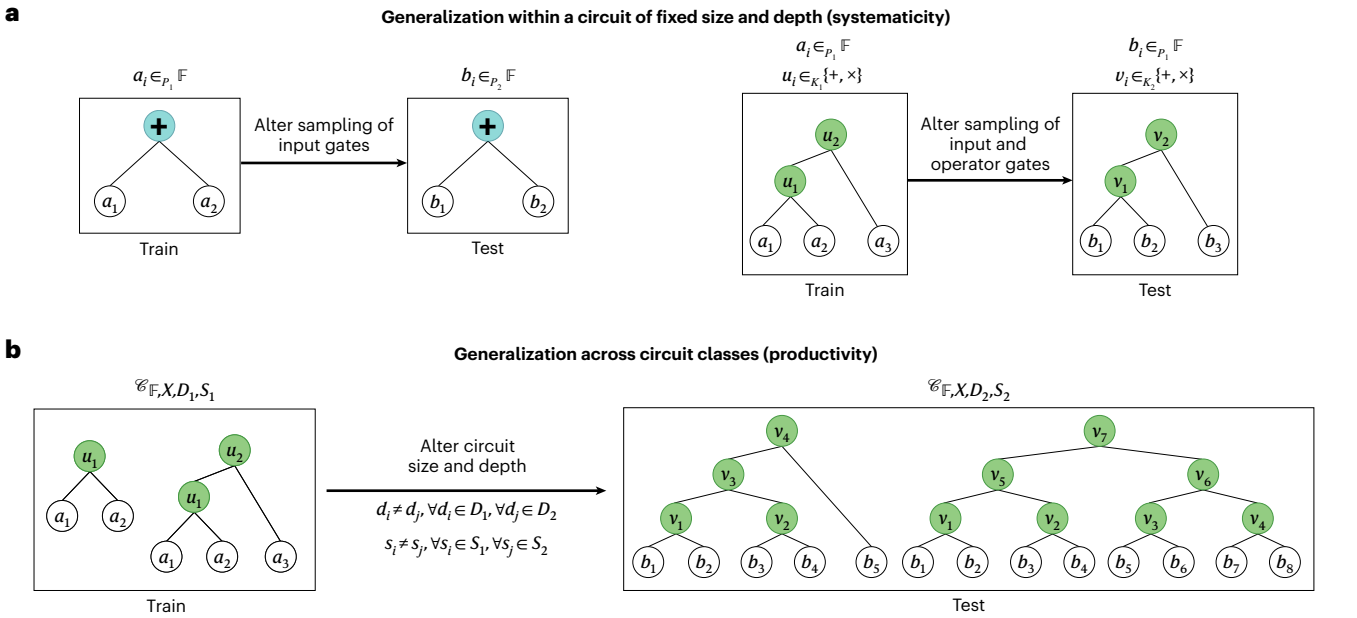


**Fig. 4 | Analogues of common compositional generalization benchmarks in terms of algebraic circuits. a**, A simpler form of generalization within an algebraic circuit is generalizing to circuits of the same structure (size and depth), but with a novel combination of gates. The analogue of this in compositionality is commonly referred to as systematic compositional generalization[9]. (Learning over circuits of fixed structure can also be viewed as learning a regression model[55].) On the left, we illustrate an example of a model that is trained on a restricted family of circuits where the input gates are sampled from a field $\mathbb{F}$ with a sampling function $P_1$. At test time, the model is required to generalize to circuits of the same circuit class (in terms of size and depth), but where the input gates are sampled using a different sampler $P_2$. On the right, both input and operator gates are chosen with separate samplers across train and test circuits, resulting in a more difficult test of systematic compositionality. **b**, Productive compositional generalization considers partitions of the training and testing sets across circuit sizes and depths. The analogue of this in compositionality is commonly referred to as productive compositional generalization. Given a family of circuits $\mathcal{C}_{\mathbb{F},X,D,S}$, where $D$ and $S$ denote a set of depths $d_i$ and size $s_i$, the primary experimental manipulation is to construct a training set $\mathcal{C}_{\mathbb{F},X,D_1,S_1}$ and testing set $\mathcal{C}_{\mathbb{F},X,D_2,S_2}$ such that there exists no overlap of a specific circuit class $C_{\mathbb{F},X,d_i,s_i}$ between the training and testing sets.

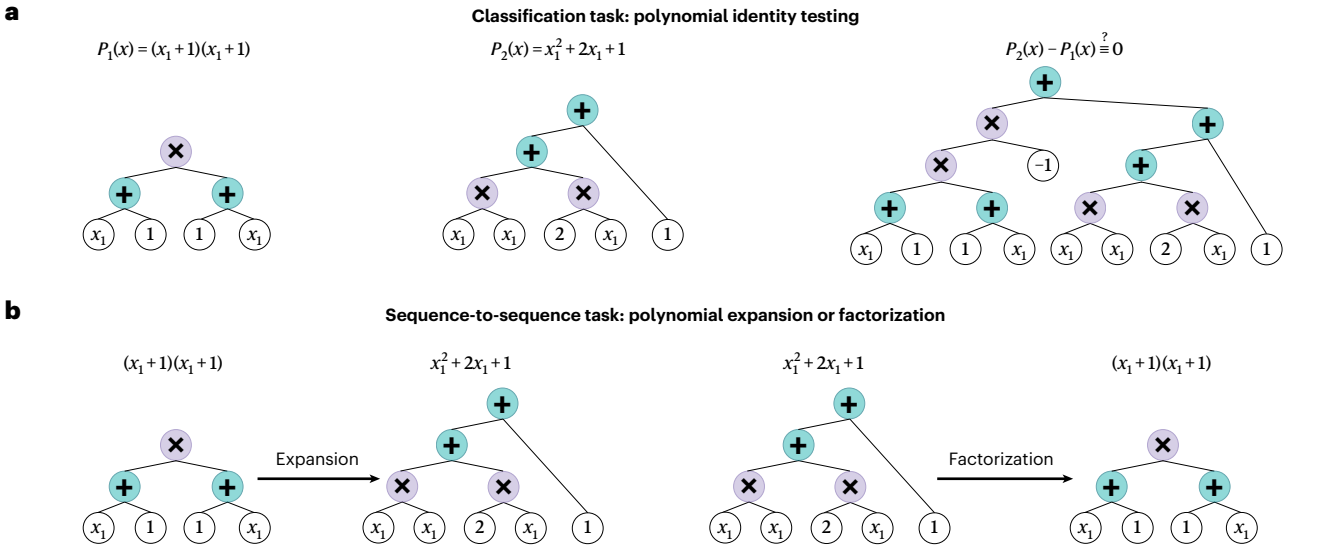


**Fig. 5 | Algebraic problems as machine learning challenges.** Previous work that leverages arithmetic problems for machine learning studies is typically limited to evaluating expressions with field elements. We introduce problems that can be evaluated with abstract variables. **a**, Polynomial identity testing as a machine learning classification task. Polynomial identity testing, an important and active area of research in computational algebra, evaluates whether two polynomials are equivalent. We illustrate two different polynomial expressions, $P_1(x)$ and $P_2(x)$ with distinct circuit representations, yet are mathematically equivalent. To reformulate this as a classification task for machine learning studies, one can ask whether $P_2(x) - P_1(x) \equiv 0$ (right). **b**, Polynomial expansion and/or factorization as a transduction (sequence to sequence) task. Common sequence-to-sequence tasks in linguistics ask whether a model can expand a string using a set of rules or a grammar, such as the AI benchmark tasks SCAN or PCFG[8,9]. In algebraic circuits, this is analogous to expanding a polynomial in a factorized representation (left). Given the one-to-one correspondence of polynomials, an additional approach is to take a polynomial in its expanded form (sum of monomials), and generate the factorized representation (right). This provides the ability to evaluate whether an AI system can expand an encoding (expansion) or compress an encoding (factorization).

https://doi.org/10.1038/s42256-025-01092-w

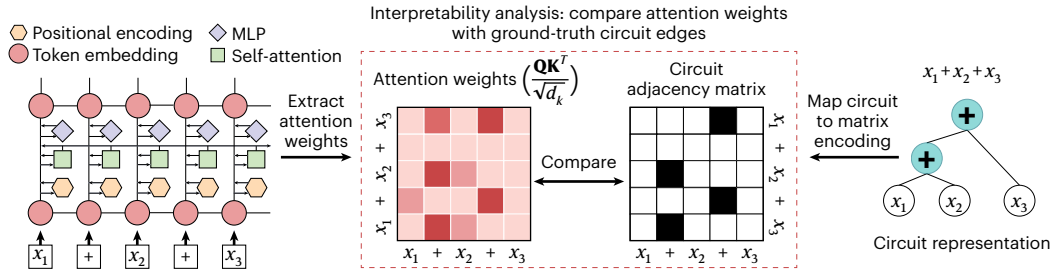


**Fig. 6 | Using an algebraic circuit's adjacency matrix as a ground-truth comparison to interpret transformer attention representations.** Left: the transformer's attention mechanism provides a useful way to peer into the representations of its input tokens. When the input is an algebraic expression (presented as a string of tokens), the attention matrix can be investigated to uncover the relationships between tokens (that is, operators and operands). Right: however, an algebraic expression can always be mapped to a circuit encoding, which can be represented as the adjacency matrix of that circuit. Middle: this allows for the direct comparison between attention weights (that is, the dot product between the query (**Q**) and transposed key (**K**) matrix) and the ground-truth circuit representation. This distance between the transformer's attention weights and the ground-truth circuit adjacency matrix can also be used as a regularizer to encourage learning circuit algorithms via attention weights. MLP, multilayer perceptron.

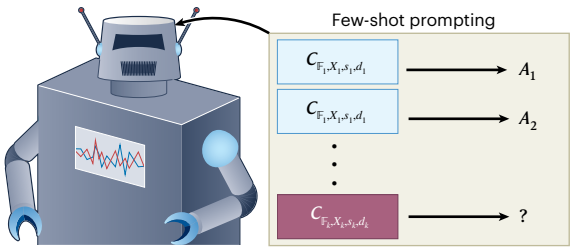


**Fig. 7 | The algebraic generalization capability across circuit divergence metrics can be evaluated through few-shot prompting in LLMs.** Given a set of question (left) and answer (right, $A_i$) pairs sampled from a specific circuit class $C_{\mathbb{F}_1,x_1,s_1,d_1}$ as prompts, we can ask an LLM to what degree it can generalize to algebraic problems sampled from a different circuit class $C_{\mathbb{F}_k,x_k,s_k,d_k}$.

Other studies have investigated the theoretical requirements for transformer models to evaluate formal languages and algorithms[48,69,82–84]. While these studies provide useful insight into what architectural components are likely important to implement algorithmic problems of a particular complexity (for example, deeper networks for circuit depth and wider context windows for circuit size), these studies do not address the learnability of circuit algorithms. Furthermore, it will be interesting to understand how frameworks grounded in algorithmic circuit complexity (as presented here) relate to other measures of compositionality, which are often measured by properties of the computation graph of a task[24], divergence of input–output mapping of a task[46], or divergence of a distribution of training and testing tasks (for example, MCD)[17].

Nevertheless, given the lack of any framework to measure algorithmic complexity in AI, we believe that introducing a circuit complexity framework from which to design quantitative benchmarks will be an important step towards building a science of algorithmic generalization. Furthermore, an algebraic circuit approach offers extensive machinery to enable generalization through other algebraic tools, such as minimum spanning/basis sets, their decompositions and more.

## Faithful algorithmic representation learning

Although an algebraic circuit encodes an explicit algorithm to compute a polynomial at a particular level of abstraction (that is, follow the edges from the input gates), it is possible that there are alternative viable algorithms to compute that same polynomial. For example, would an AI model simply follow the edges from the input gates to the output gates? Or might it factorize the expression (leading to a shallower circuit) before evaluating that expression? Relatedly, the algorithmic steps required to implement long addition (particularly when computing the sum of two very large numbers) are not fully captured in the algebraic circuits we present in this Perspective. (For example, long addition as specified in Dziri et al.[24] requires additional operators such as 'carry', 'concatenate', 'modulo' and so on.) Would AI models implement long addition with a different set of computational gates and operators to accommodate arithmetic with large numbers? Nevertheless, despite these potential ambiguities, using a computational circuit framework provides testable and verifiable hypotheses that allow us to empirically evaluate what algorithm a model implements. Furthermore, use of a circuit framework enables the design of quantitatively meaningful algorithmic benchmarks such as those designed to test generalization over algorithmic time complexity (circuit depth) or space complexity (circuit size), among others. More broadly, a computational circuit framework can naturally extend beyond algebraic circuits to formal languages, such as those within the Chomsky hierarchy[47] (for example, context-free grammars), by characterizing the circuit parameters of the language's parse tree. Thus, this framework provides a unified theoretical foundation for quantifying algorithmic complexity across diverse empirical evaluations, including arithmetic tasks, formal languages and other compositional paradigms, and is an important step towards understanding the faithfulness by which an AI system computes a class of algorithms.

Alternatively, many previous approaches to faithfully learning algorithmic representations often involve neurosymbolic methods. These methods provide promising avenues to learn discrete algorithmic solutions to problems that are reliable and sample efficient[53,85–88]. However, designing general purpose (rather than domain specific) neurosymbolic models can be challenging, as they are often not fully differentiable or require strong inductive biases. By contrast, although statistical machine learning models (for example, transformers) are 'general purpose', the learning process is often obfuscated by learning dynamics that depend on architecture and complex optimization protocols. This makes it difficult to ascertain what algorithms statistical systems learn. However, recent studies in compositional representation learning have suggested that different factors—such as choice of initialization and/or training curriculum—can have a strong influence on whether a model learns compositionally[49,89–92]. In addition, developing techniques from the field of mechanistic interpretability provides new avenues from which to inspect whether the learned representations in an AI model are faithful to the hypothesized underlying algorithm (for example, Fig. 6; Olsson et al.[64] and Friedman et al.[93]). Nevertheless, leveraging diverse methods to carefully investigate the algorithms that symbolic and statistical AI models learn will be important for their interpretability, reliability and overall safety to ensure reliable deployment of AI systems.

## Conclusion

Quantifying the algorithmic ability of AI systems is difficult owing to the lack of a theoretical framework from which to establish meaningful benchmarks. While there has been an increasing number of studies that have employed algebraic and compositional tasks to reliably elicit failure modes of transformers and LLMs, no theoretical framework exists to interpret these findings. In this Perspective, we provide a parsimonious framework—algebraic circuit complexity—to evaluate the extent of a model's algorithmic generalization ability in terms of their circuit divergences. In contrast to other formulations of complexity, such as Kolmogorov complexity, encoding algorithmic problems as circuits provides an explicitly computable formulation. The rich expressivity of algebraic problems, the data-rich nature of producing algebraic datasets and the close links with algebraic circuits to other mathematical fields, makes algebraic circuit complexity a fruitful approach to quantify algorithmic generalization in AI systems. More generally, the circuit complexity framework introduced here (with algebraic functions) can be naturally extended to other computational problems (for example, formal languages, Boolean circuits). We hope this Perspective provides the theoretical groundwork for future studies to quantify algorithmic generalization in modern AI systems with circuits.

## References


1. Bubeck, S. et al. Sparks of artificial general intelligence: early experiments with GPT-4. Preprint at https://arxiv.org/abs/2303.12712 (2023).
2. Wei, J. et al. Emergent abilities of large language models. *Trans. Mach. Learn. Res.* (2022); https://openreview.net/pdf?id=yzkSU5zdwD
3. Webb, T., Holyoak, K. J. & Lu, H. Emergent analogical reasoning in large language models. *Nat. Hum. Behav.* https://doi.org/10.1038/s41562-023-01659-w (2023).
4. DeepSeek-AI et al. DeepSeek-R1: incentivizing reasoning capability in LLMs via reinforcement learning. Preprint at https://arxiv.org/abs/2501.12948 (2025).
5. Kim, N., Linzen, T. & Smolensky, P. Uncontrolled lexical exposure leads to overestimation of compositional generalization in pretrained models. Preprint at https://arxiv.org/abs/2212.10769 (2022).
6. Schaeffer, R., Miranda, B. & Koyejo, S. Are emergent abilities of large language models a mirage? *Adv. Neural Inf. Process. Syst.* **36**, 55565–55581 (2023).
7. Wu, Z. et al. Reasoning or reciting? Exploring the capabilities and limitations of language models through counterfactual tasks. In *Proc. 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)* (eds Duh, K. et al.) 1819–1862 (Association for Computational Linguistics, 2024); https://doi.org/10.18653/v1/2024.naacl-long.102
8. Lake, B. & Baroni, M. Generalization without systematicity: on the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning* 2873–2882 (PMLR, 2018); http://proceedings.mlr.press/v80/lake18a.html
9. Hupkes, D., Dankers, V., Mul, M. & Bruni, E. Compositionality decomposed: how do neural networks generalise? *J. Artif. Intell. Res.* **67**, 757–795 (2020).
10. Hudson, D. A. & Manning, C. D. GQA: a new dataset for real-world visual reasoning and compositional question answering. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* 6700–6709 (IEEE, 2019); https://openaccess.thecvf.com/content_CVPR_2019/html/Hudson_GQA_A_New_Dataset_for_Real-World_Visual_Reasoning_and_Compositional_CVPR_2019_paper.html
11. Yang, G. R., Ganichev, I., Wang, X.-J., Shlens, J. & Sussillo, D. A dataset and architecture for visual reasoning with a working memory. In *Computer Vision – ECCV 2018* Lecture Notes in Computer Science Vol. 11214 (eds Ferrari, V. et al.) 729–745 (Springer, 2018); https://doi.org/10.1007/978-3-030-01249-6_44
12. Ito, T., Dan, S., Rigotti, M., Kozloski, J. & Campbell, M. On the generalization capacity of neural networks during generic multimodal reasoning. In *International Conference on Learning Representations* (2024); https://openreview.net/forum?id=zyBJodMrn5&noteId=zyBJodMrn5
13. Johnson, J. et al. CLEVR: a diagnostic dataset for compositional language and elementary visual reasoning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 1988–1997 (IEEE, 2017); https://doi.org/10.1109/CVPR.2017.215
14. Clark, C. et al. BoolQ: exploring the surprising difficulty of natural yes/no questions. In *Proc. 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (eds Burstein, J. et al.) 2924–2936 (Association for Computational Linguistics, 2019); https://doi.org/10.18653/v1/N19-1300
15. Kim, N. & Linzen, T. COGS: a compositional generalization challenge based on semantic interpretation. In *Proc. 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (eds Webber, B. et al.) 9087–9105 (Association for Computational Linguistics, 2020); https://doi.org/10.18653/v1/2020.emnlp-main.731
16. Ruis, L., Andreas, J., Baroni, M., Bouchacourt, D. & Lake, B. M. A benchmark for systematic generalization in grounded language understanding. *Adv. Neural Inf. Process. Syst.* **33**, 19861–19872 (2020).
17. Keysers, D. et al. Measuring compositional generalization: a comprehensive method on realistic data. In *International Conference on Learning Representations* (2020); https://openreview.net/forum?id=SygcCnNKwr&utm_campaign=Graph
18. Kudo, K. et al. Do deep neural networks capture compositionality in arithmetic reasoning? In *Proc. 17th Conference of the European Chapter of the Association for Computational Linguistics* (Vlachos, A. & Augenstein, I.) 1351–1362 (Association for Computational Linguistics, 2023); https://doi.org/10.18653/v1/2023.eacl-main.98
19. McLeish, S. et al. Transformers can do arithmetic with the right embeddings. *Adv. Neural Inf. Proc. Syst.* **37**, 108012–108041 (2024).
20. Zhou, H. et al. What algorithms can transformers learn? A study in length generalization. In *International Conference on Learning Representations* (2024); https://openreview.net/forum?id=AssIuHnmHX
21. Shen, R. et al. Positional description matters for transformers arithmetic. In *International Conference on Learning Representations* (2024); https://openreview.net/forum?id=ZMuPAOY8Oz
22. Saxton, D., Grefenstette, E., Hill, F. & Kohli, P. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations* (2019); https://openreview.net/forum?id=H1gR5iR5FX
23. Lee, N., Sreenivasan, K., Lee, J. D., Lee, K. & Papailiopoulos, D. Teaching arithmetic to small transformers. In *International Conference on Learning Representations* (2023); https://openreview.net/forum?id=dsUB4bst9S
24. Dziri, N. et al. Faith and fate: limits of transformers on compositionality. *Adv. Neural Inf. Process. Syst.* **36**, 70293–70332 (2023).
25. McCoy, R. T., Yao, S., Friedman, D., Hardy, M. & Griffiths, T. L. Embers of autoregression show how large language models are shaped by the problem they are trained to solve. *Proc. Natl Acad. Sci. USA* **121**, e2322420121 (2024).

26. Zhou, Y. et al. Transformers can achieve length generalization but not robustly. In *International Conference on Learning Representations* (2024); https://openreview.net/forum?id=DWkWIh3vFJ
27. Sinha, S., Premsri, T. & Kordjamshidi, P. A survey on compositional learning of AI models: theoretical and experimental practices. *Trans. Mach. Learn. Res.* https://openreview.net/forum?id=BXDxwItNqQ (2024).
28. Frege, G. in *Logic and Philosophy for Linguists* (ed. Moravcsik, J. M. E.) 279–298 (De Gruyter, 1975); https://doi.org/10.1515/9783111546216-018
29. Carnap, R. *Meaning and Necessity: A Study in Semantics and Modal Logic* Vol. 30 (Univ. Chicago Press, 1988).
30. Russin, J., McGrath, S. W., Williams, D. J. & Elber-Dorozko, L. From Frege to chatGPT: compositionality in language, cognition, and deep neural networks. Preprint at https://arxiv.org/abs/2405.15164 (2024).
31. Kazemnejad, A., Padhi, I., Natesan Ramamurthy, K., Das, P. & Reddy, S. The impact of positional encoding on length generalization in transformers. *Ad. Neural Inf. Process. Syst.* **36**, 24892–24928 (2023).
32. Hupkes, D. et al. A taxonomy and review of generalization research in NLP. *Nat. Mach. Intell.* **5**, 1161–1174 (2023).
33. Fodor, J. A. & Pylyshyn, Z. W. Connectionism and cognitive architecture: a critical analysis. *Cognition* **28**, 3–71 (1988).
34. Fodor, J. & McLaughlin, B. P. Connectionism and the problem of systematicity: why Smolensky's solution doesn't work. *Cognition* **35**, 183–204 (1990).
35. Smolensky, P. in *Connectionism and the Philosophy of Mind* (eds Horgan, T. & Tienson, J.) 281–308 (Springer, 1991); https://doi.org/10.1007/978-94-011-3524-5_13
36. Deletang, G. et al. Neural networks and the chomsky hierarchy. In *International Conference on Learning Representations* (2022); https://openreview.net/forum?id=WbxHAzkeQcn
37. Zhou, H. et al. Teaching algorithmic reasoning via in-context learning. In *International Conference on Learning Representations* (2023); https://openreview.net/forum?id=6dlC7E1H_9
38. Ruoss, A. et al. Randomized positional encodings boost length generalization of transformers. In *Proc. 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (eds Rogers, A. et al.) 1889–1903 (Association for Computational Linguistics, 2023); https://doi.org/10.18653/v1/2023.acl-short.161
39. Jelassi, S. et al. Length generalization in arithmetic transformers. Preprint at https://arxiv.org/abs/2306.15400 (2023).
40. Wang, C., Zheng, B., Niu, Y. & Zhang, Y. in *Natural Language Processing and Chinese Computing* (eds Wang, L. et al.) 758–769 (Springer, 2021); https://doi.org/10.1007/978-3-030-88480-2_61
41. Nogueira, R., Jiang, Z. & Lin, J. Investigating the limitations of transformers with simple arithmetic tasks. Preprint at https://arxiv.org/abs/2102.13019 (2021).
42. Shpilka, A. & Yehudayoff, A. Arithmetic circuits: a survey of recent results and open questions. *Found. Trends Theor. Comput. Sci.* **5**, 207–388 (2010).
43. Bürgisser, P., Clausen, M. & Shokrollahi, M. A. *Algebraic Complexity Theory* Vol. 315 (Springer Science & Business Media, 2013).
44. Arora, S. & Barak, B. *Computational Complexity: A Modern Approach* (Cambridge Univ. Press, 2009).
45. Wyeth, C. & Sturtivant, C. A circuit complexity formulation of algorithmic information theory. *Physica D* **456**, 133925 (2023).
46. Ram, P., Klinger, T. & Gray, A. G. What makes models compositional? A theoretical view: with supplement. Preprint at https://arxiv.org/abs/2405.02350 (2024).
47. Chomsky, N. Three models for the description of language. *IRE Trans. Inf. Theory* **2**, 113–124 (1956).
48. Strobl, L., Merrill, W., Weiss, G., Chiang, D. & Angluin, D. What formal languages can transformers express? A survey. *Trans. Assoc. Comput. Linguist.* **12**, 543–561 (2024).
49. Lake, B. M. & Baroni, M. Human-like systematic generalization through a meta-learning neural network. *Nature* https://doi.org/10.1038/s41586-023-06668-3 (2023).
50. Ruis, L. & Lake, B. Improving systematic generalization through modularity and augmentation. Preprint at https://arxiv.org/abs/2202.10745 (2022).
51. Ontanon, S., Ainslie, J., Fisher, Z. & Cvicek, V. Making transformers solve compositional tasks. In *Proc. 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (eds Muresan, S. et al) 3591–3607 (Association for Computational Linguistics, 2022); https://doi.org/10.18653/v1/2022.acl-long.251
52. Csordás, R., Irie, K. & Schmidhuber, J. The devil is in the detail: simple tricks improve systematic generalization of transformers. In *Proc. 2021 Conference on Empirical Methods in Natural Language Processing* (eds Moens, M.-F. et al.) 619–634 (Association for Computational Linguistics, 2021); https://doi.org/10.18653/v1/2021.emnlp-main.49
53. Klinger, T. et al. Compositional program generation for systematic generalization. In *Proc. 2021 Conference on Empirical Methods in Natural Language Processing* (2023); https://openreview.net/forum?id=Wxj9U0ySU-s
54. Poggio, T. & Fraser, M. Compositional sparsity of learnable functions. article, center for brains, minds and machines (CBMM). *MIT Libraries* https://dspace.mit.edu/handle/1721.1/153475 (2024).
55. Zhang, X., Blanchet, J., Ghosh, S. & Squillante, M. S. A class of geometric structures in transfer learning: minimax bounds and optimality. In *Proc. 25th International Conference on Artificial Intelligence and Statistics* 3794–3820 (PMLR, 2022); https://proceedings.mlr.press/v151/zhang22a.html
56. Ghosh, S., Squillante, M. & Wollega, E. Efficient generalization with distributionally robust learning. *Adv. Neural Inf. Process. Syst.* **34**, 28310–28322 (2021).
57. Hadley, R. F. Systematicity in connectionist language learning. *Mind Lang.* **9**, 247–272 (1994).
58. Meurer, A. et al. SymPy: symbolic computing in Python. *PeerJ Comput. Sci.* **3**, e103 (2017).
59. Hendrycks, D. et al. Measuring massive multitask language understanding. In *International Conference on Learning Representations* (2020); https://openreview.net/forum?id=d7KBjmI3GmQ
60. Andreas, J. Measuring compositionality in representation learning. In *International Conference on Learning Representations* (2018); https://openreview.net/forum?id=HJz05o0qK7
61. Nezhurina, M., Cipolina-Kun, L., Cherti, M. & Jitsev, J. Alice in Wonderland: simple tasks showing complete reasoning breakdown in state-of-the-art large language models. Preprint at https://arxiv.org/abs/2406.02061 (2024).
62. Chan, S. C. Y. et al. Transformers generalize differently from information stored in context vs in weights. Preprint at https://arxiv.org/abs/2210.05675 (2022).
63. Reddy, G. The mechanistic basis of data dependence and abrupt learning in an in-context classification task. In *International Conference on Learning Representations* (2024); https://openreview.net/forum?id=aN4Jf6Cx69
64. Olsson, C. et al. In-context learning and induction heads. Preprint at https://arxiv.org/abs/2209.11895 (2022).
65. Zhou, D. et al. Least-to-most prompting enables complex reasoning in large language models. In *International Conference on Learning Representations* (2022); https://openreview.net/forum?id=WZH7099tgfM

66. Wei, J. et al. Chain-of-thought prompting elicits reasoning in large language models. *Adv. Neural Inf. Process. Syst.* **35**, 24824–24837 (2022).
67. Nye, M. et al. Show your work: scratchpads for intermediate computation with language models. *OpenReview.net* https://openreview.net/forum?id=iedYJm92o0a&ref=morioh.com&utm_source=morioh.com (2021).
68. Feng, G. et al. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Adv. Neural Inf. Process. Syst.* **36**, 70757–70798 (2023).
69. Merrill, W. & Sabharwal, A. The expressive power of transformers with chain of thought. In *International Conference on Learning Representations* (2024); https://openreview.net/forum?id=NjNGlPh8Wh
70. Li, Z., Liu, H., Zhou, D. & Ma, T. Chain of thought empowers transformers to solve inherently serial problems. In *International Conference on Learning Representations* (2024); https://openreview.net/forum?id=3EWTEy9MTM
71. Shao, Z. et al. DeepSeekMath: pushing the limits of mathematical reasoning in open language models. Preprint at https://arxiv.org/abs/2402.03300 (2024).
72. Mitchell, M., Palmarini, A. B. & Moskvichev, A. Comparing humans, GPT-4, and GPT-4V on abstraction and reasoning tasks. Preprint at https://arxiv.org/abs/2311.09247 (2023).
73. LeGris, S., Vong, W. K., Lake, B. M. & Gureckis, T. M. H-ARC: a robust estimate of human performance on the abstraction and reasoning corpus benchmark. Preprint at https://arxiv.org/abs/2409.01374 (2024).
74. McClelland, J. L. et al. Letting structure emerge: connectionist and dynamical systems approaches to cognition. *Trends Cogn. Sci.* **14**, 348–356 (2010).
75. Chollet, F. On the measure of intelligence. Preprint at https://arxiv.org/abs/1911.01547 (2019).
76. Fedor, A., Varga, M. & Szathmáry, E. Semantics boosts syntax in artificial grammar learning tasks with recursion. *J. Exp. Psychol. Learn. Mem. Cogn.* **38**, 776–782 (2012).
77. Lampinen, A. K. et al. Language models, like humans, show content effects on reasoning tasks. *PNAS Nexus* **3**, 233 (2024).
78. Kolmogorov, A. N. Three approaches to the quantitative definition of information. *Int. J. Comput. Math.* **2**, 157–168 (1968).
79. Li, M. & Vitányi, P. *An Introduction to Kolmogorov Complexity and Its Applications* Texts in Computer Science (Springer, 2019).
80. Johnston, I. G. et al. Symmetry and simplicity spontaneously emerge from the algorithmic nature of evolution. *Proc. Natl Acad. Sci. USA* **119**, e2113883119 (2022).
81. Dingle, K., Camargo, C. Q. & Louis, A. A. Input–output maps are strongly biased towards simple outputs. *Nat. Commun.* **9**, 761 (2018).
82. Merrill, W. & Sabharwal, A. A little depth goes a long way: the expressive power of log-depth transformers. Preprint at https://arxiv.org/abs/2503.03961 (2025).
83. Yang, A., Chiang, D. & Angluin, D. Masked hard-attention transformers recognize exactly the star-free languages. *Adv. Neural Inf. Proc. Syst.* **37**, 10202–10235 (2024).
84. Amiri, A., Huang, X., Rofin, M. & Hahn, M. Lower bounds for chain-of-thought reasoning in hard-attention transformers. Preprint at https://arxiv.org/abs/2502.02393 (2025).
85. Lake, B. M., Salakhutdinov, R. & Tenenbaum, J. B. Human-level concept learning through probabilistic program induction. *Science* **350**, 1332–1338 (2015).
86. Poesia, G. & Goodman, N. D. Peano: learning formal mathematical reasoning. *Phil. Trans. R. Soc. A* **381**, 20220044 (2023).
87. Trinh, T. H., Wu, Y., Le, Q. V., He, H. & Luong, T. Solving olympiad geometry without human demonstrations. *Nature* **625**, 476–482 (2024).
88. Ellis, K. et al. DreamCoder: bootstrapping inductive program synthesis with wake–sleep library learning. In *Proc. 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* 835–850 (Association for Computing Machinery, 2021); https://doi.org/10.1145/3453483.3454080
89. Lippl, S. & Stachenfeld, K. When does compositional structure yield compositional generalization? A kernel theory. Preprint at https://arxiv.org/abs/2405.16391 (2024).
90. Ito, T. et al. Compositional generalization through abstract representations in human and artificial neural networks. *Adv. Neural Inf. Process. Syst.* **35**, 32225–32239 (2022).
91. Zhang, Z., Lin, P., Wang, Z., Zhang, Y. & Xu, Z.-Q. J. Initialization is critical to whether transformers fit composite functions by reasoning or memorizing. *Adv. Neural Inf. Proc. Syst.* **37**, 14093–14126 (2024).
92. Saglietti, L., Mannelli, S. & Saxe, A. An analytical theory of curriculum learning in teacher–student networks. *Adv. Neural Inf. Process. Syst.* **35**, 21113–21127 (2022).
93. Friedman, D., Wettig, A. & Chen, D. Learning transformer programs. *Adv. Neural Inf. Process. Syst.* **36**, 49044–49067 (2023).

## Acknowledgements

We thank M. Carmosino and K. Srivastava for helpful discussions on earlier versions of the paper. We acknowledge funding support from the Exploratory Science Councils at IBM Research.

## Competing interests

The authors declare no competing interests.

## Additional information

**Correspondence** should be addressed to Takuya Ito.

**Peer review information** *Nature Machine Intelligence* thanks Martha Lewis, and the other, anonymous, reviewer(s) for their contribution to the peer review of this work.