CausalARC: Abstract Reasoning with Causal World Models

Jacqueline Maasch
Cornell Tech, New York, NY

John Kalantari
YRIKKA, New York, NY

Kia Khezeli
YRIKKA, New York, NY

Abstract

Reasoning requires adaptation to novel problem settings under limited data and distribution shift. This work introduces CausalARC: an experimental testbed for AI reasoning in low-data and out-of-distribution regimes, modeled after the Abstraction and Reasoning Corpus (ARC). Each CausalARC reasoning task is sampled from a fully specified *causal world model*, formally expressed as a structural causal model. Principled data augmentations provide observational, interventional, and counterfactual feedback about the world model in the form of few-shot, in-context learning demonstrations. As a proof-of-concept, we illustrate the use of CausalARC for four language model evaluation settings: (1) abstract reasoning with test-time training, (2) counterfactual reasoning with in-context learning, (3) program synthesis, and (4) causal discovery with logical reasoning.

1 Introduction

Humans are exceptional few-shot learners that use internal representations of the world to navigate novel scenarios [12, 20]. These internal *world models* can enable general reasoning, for which generalization is both robust under "known unknowns" and flexible under "unknown unknowns" [4]. World models can encode beliefs about cause-effect relationships, supporting general reasoning at all three levels of the Pearl Causal Hierarchy (PCH; Figure 1) [2]. Though causal reasoning is a hallmark of human cognition [9] and a desideratum for human-like AI [32, 19], state-of-the-art generative models do not yet display general reasoning at all three levels of the PCH [17, 11, 45, 37, 16, 25].

Despite the emergence of few-shot [3] and in-context learning [43] in language models (LMs), general reasoning still poses a major distribution shift challenge. One promising direction for out-of-distribution problem-solving is *test-time training* (TTT) [40]: a strategy where model parameters are temporarily updated on each test instance (or batch) for on-the-fly adaptation to problems outside the pretraining distribution [30, 24]. Testament to its utility for reasoning out-of-distribution, TTT was the dominant strategy in winning submissions for the first Abstraction and Reasoning Corpus (ARC) [4, 5, 23, 1]: the premier benchmark for general intelligence in AI to-date. ARC is a grid world of highly diverse abstract reasoning tasks that are generally solvable by humans. Each task is governed by a unique rule that deterministically transforms input arrays to output arrays, as demonstrated by few-shot learning examples. Ablation studies show that fine-tuning plus TTT with few-shot, in-context learning can significantly improve LM performance on ARC relative to base models [1].

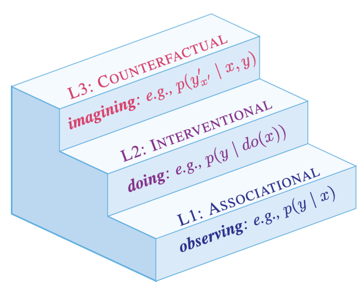

Figure 1: The PCH: observing factual realities (L1), exerting actions to induce interventional realities (L2), and imagining alternate counterfactual realities (L3) [2].

This work extends and reconceptualizes the ARC setup to support causal reasoning evaluation under limited data and distribution shift. We introduce CausalARC: an experimental testbed of ARC-

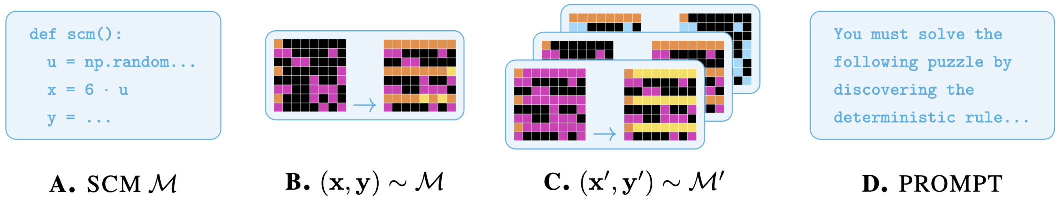

Figure 2: The CausalARC testbed. (**A**) First, SCM $\mathcal{M}$ is manually transcribed in Python code. (**B**) Input-output pairs are randomly sampled, providing observational (L1) learning signals about the world model. (**C**) Sampling from interventional submodels $\mathcal{M}'$ of $\mathcal{M}$ yields interventional (L2) samples $(\mathbf{x}', \mathbf{y}')$. Given pair $(\mathbf{x}, \mathbf{y})$, performing multiple interventions while holding the exogenous context constant yields a set of counterfactual (L3) pairs. (**D**) Using L1 and L3 pairs as in-context demonstrations, we can automatically generate natural language prompts for diverse reasoning tasks.

like reasoning tasks sampled from fully specified *causal world models* (Figure 2). World models are expressed as probabilistic *structural causal models* (SCMs), a rich formalism for representing data generating processes and simulating hypothetical outcomes [32, 33]. Given a fully specified SCM, all three levels of the PCH are well-defined: any observational (L1), interventional (L2), or counterfactual (L3) query can be answered about the environment under study [2]. This formulation makes CausalARC an open-ended playground for testing reasoning hypotheses at all three levels of the PCH, with an emphasis on abstract, logical, and counterfactual reasoning.

Design CausalARC is designed to address multiple core problems in AI reasoning evaluation:

- *Reasoning tasks as generative models.* CausalARC frames each task as a generative model (SCM) from which task instances can be randomly sampled. Reasoning robustness can be benchmarked with respect to a distribution of task instances from each world model, enabling uncertainty estimation. Generative benchmarks that randomly sample from world models can also buffer against static benchmark data leakage, a major challenge in AI evaluation [27, 44, 25, 10].
- *Disentangling signal from noise.* Differentiating true reasoning from statistical recall is a nontrivial objective in AI evaluation [16, 44, 27, 36]. ARC-like settings are conducive to this objective, as abstract reasoning relies more on innate cognitive priors than factual knowledge [4].
- *Random sampling at all levels of the PCH*. Unlike prior ARC extensions, the user can sample L1, L2, and L3 distributions of any sample size from a given world model. This feature also facilitates causal discovery and causal inference queries that require a sample distribution.
- *Fine-grained error analyses.* CausalARC is designed for hypothesis-driven model evaluations and detailed error analyses. As in ConceptARC [29], tasks are labeled by reasoning theme (logical, counting, extension, and ordering). As in Shojaee et al. [36], CausalARC supports evaluation with respect to scaling problem complexity by configuring most tasks with tunable array sizes.

Intended Use Cases CausalARC is a richly annotated experimental testbed for exploring diverse AI reasoning hypotheses. Unlike ARC-Heavy [23], ARC-Potpourri [23], and RE-ARC [14], CausalARC is not intended to be a large-scale fine-tuning dataset. We avoid the automated data augmentations used in Li et al. [23], Hodel [14], and Akyürek et al. [1] to ensure that data meet the stringent assumptions of the causal framework that we describe. Applications for CausalARC could include: benchmarking abstract reasoning in LMs, as in Chollet et al. [5]; benchmarking causal inference and causal discovery abilities, as in Jin et al. [17] and Jin et al. [18]; probing impacts of prompt formulation on in-context learning and TTT; benchmarking reasoning with respect to parametric assumptions on the SCM, topological properties of the causal graph, or scaling task complexity; etc.

Contributions

§3 *The CausalARC testbed*. We introduce an open-ended experimental testbed for AI reasoning at all three levels of the PCH. We provide a static dataset and a public codebase for task generation.[1]

§4 *Empirical demonstrations.* As a proof-of-concept, we illustrate the use of CausalARC for four LM evaluation settings: (1) abstract reasoning with TTT, (2) counterfactual reasoning with in-context learning, (3) program synthesis, and (4) causal discovery with logical reasoning. Preliminary comparisons to ARC-AGI-1 suggest that CausalARC is of similar difficulty.

[1] https://anonymous.4open.science/r/causal_arc-E237/. Hugging Face and GitHub provided at camera-ready.

2 Preliminaries

We briefly outline preliminaries on causal modeling and the ARC benchmark. Appendix B contains extended preliminaries on world models, intelligence, generalization, and reasoning.

2.1 Structural Causal Models

The reasoning framework presented here draws from the theory of *structural causal models* [7]. Moving forward, we denote random variables with uppercase letters, sets of random variables with bold letters, and realizations in lowercase (e.g., $V = v$, $\mathbf{V} = \mathbf{v}$).

Definition 2.1 (Structural causal model (SCM), Bareinboim et al. [2])**.** An SCM is a tuple $\mathcal{M} := \langle \mathbf{U}, p(\mathbf{u}), \mathbf{V}, \mathcal{F} \rangle$ where $\mathbf{U} = \{U_i\}_{i=1}^m$ is the set of exogenous variables explained by mechanisms external to $\mathcal{M}$, $p(\mathbf{u})$ is the distribution over $\mathbf{U}$, $\mathbf{V} = \{V_i\}_{i=1}^n$ is the set of endogenous variables explained by variables in $\mathbf{U} \cup \mathbf{V}$, and $\mathcal{F} = \{f_i\}_{i=1}^n$ is the set of structural functions such that $v_i = f_i(\mathbf{pa}_{v_i}, \mathbf{u}_i)$ for endogenous parent set $\mathbf{pa}_{v_i}$ and exogenous context $\mathbf{u}_i$.

The joint factorization of $p(\mathbf{v}, \mathbf{u})$ implied by $\mathcal{M}$ can be represented as a causal graph $\mathcal{G}$. This work assumes that $\mathcal{G}$ is a directed acyclic graph (DAG), justified by the fact that our testbed does not have a time series component (and thus no feedback loops). Additionally, we make the standard positivity assumption: an SCM is positive if $p(v_i) > 0$ for every realization $V_i \in \mathbf{V} = v_i$. We do not make assumptions about confounding. Future work could explore relaxations of these assumptions.

Crucially, we can exert actions or *interventions* on $\mathcal{M}$ to induce *interventional distributions*. In the real world, intervention corresponds to controlled experimentation.

Definition 2.2 (Hard intervention)**.** A hard intervention $do(V_i = v_i)$ replaces the true causal function $f_i(\mathbf{pa}_{v_i}, u_i)$ with the constant function evaluating to v_i.

Definition 2.3 (Soft intervention)**.** A soft intervention modifies the local conditional distribution $p(v_i \mid \mathbf{pa}_{v_i})$ to some new distribution $q(v_i \mid \mathbf{pa}'_{v_i})$, where $\mathbf{pa}'_{v_i}$ may or may not differ from $\mathbf{pa}_{v_i}$.

Next, we can define *counterfactuals* with respect to $\mathcal{M}$. In the real world, counterfactuals are not measurable through controlled experimentation as they correspond to alternate hypothetical realities.

Definition 2.4 (Counterfactual, Pearl [33])**.** Let $\mathcal{M}_x$ be the submodel of $\mathcal{M}$ induced by hard or soft intervention on $X \in \mathbf{V}$. Let $Y \in \mathbf{V}$ be a variable whose value we wish to query. The counterfactual Y_x under model $\mathcal{M}$ is then expressed as $Y_x(\mathbf{u}) := Y_{\mathcal{M}_x}(\mathbf{u})$.

That is, the counterfactual under the original SCM is equal to the value taken by Y under the interventional submodel $\mathcal{M}_x$, with the exogenous context $\mathbf{u}$ fixed. For hard interventions, $Y_x(\mathbf{u}) = y$ could be verbalized as "Y would have been y had X been x in context $\mathbf{U} = \mathbf{u}$" [33]. Thus, counterfactuals are jointly distributed random variables in a shared probability space over $\mathbf{U}$ [41].

With these definitions of interventions and counterfactuals in hand, we can define a *hierarchy of information* encoded in the distributions induced by the submodels of $\mathcal{M}$ (Figure 1).

Definition 2.5 (Pearl Causal Hierarchy (PCH), Bareinboim et al. [2])**.** Let $\mathcal{M}$ be a fully specified SCM, per Definition 2.1. The PCH is the set of all observational (layer L1), interventional (layer L2), and counterfactual (layer L3) distributions induced by $\mathcal{M}$.

Given access to the fully specified $\mathcal{M}$, the PCH is well-defined: the information encoded in $\mathcal{M}$ enables valuation for any quantity at L1, L2, and L3 [2]. Notably, lower layers of the PCH generally underdetermine higher layers: L_j encodes information sufficient to answer queries at $\text{L}_{i\leq j}$, yet knowledge of $\text{L}_{i<j}$ is *almost never* sufficient to answer queries at L_j. We refer the reader to the Causal Hierarchy Theorem (CHT) for formal proof, as presented in Bareinboim et al. [2] .

2.2 ARC-AGI: The Abstraction & Reasoning Corpus

Background The first Abstraction and Reasoning Corpus (ARC-AGI-1) was introduced in 2019 [4].[2] At the time of writing, the second ARC challenge is underway[3] and the third is in development.

[2] https://arcprize.org/arc-agi/

[3] https://www.kaggle.com/competitions/arc-prize-2025

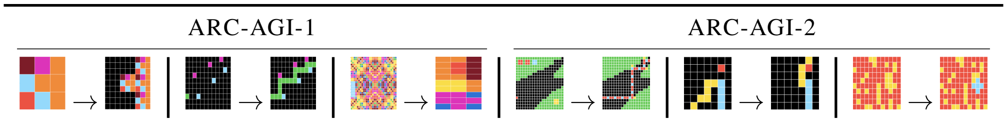

Figure 3: Example input-output pairs from ARC-AGI-1 and ARC-AGI-2.

ARC is a grid world, represented by two-dimensional arrays of pixels that can take on one of ten colors each. For ARC-AGI-1, data consist of a training set ($n = 400$ reasoning tasks), a public evaluation set ($n = 400$), a semi-private evaluation set ($n = 100$), and a fully private evaluation set ($n = 100$). Each "grid" is an array of any dimensionality from 1×1 to 30×30 (Figure 3). Arrays vary widely in appearance, at times appearing as randomly dispersed colors. Arrays can also feature distinct multi-pixel shapes, referred to as *sprites* (as in computer graphics).

The test-taker must solve each task by discovering the deterministic rule or transformation that maps input arrays to output arrays. Each task provides approximately 2–5 input-output pairs as examples to demonstrate the rule, with no additional clues provided. An average human should ostensibly be able to solve most or all tasks from these demonstrations alone, with no specialized knowledge or training. Insead, problem-solving requires innate cognitive priors, such as elementary arithmetic, basic geometry, and intuitive physics [4].

Notation Let $\mathcal{D} \coloneqq \mathcal{D}_{train} \cup \mathcal{D}_{eval}$ denote the ARC dataset. As we are concerned with in-context learning and TTT but not pretraining or fine-tuning, we consider only $\mathcal{D}_{eval}$ moving forward. Each instance in $\mathcal{D}_{eval}$ is a reasoning task $\mathbf{T}_i$, such that $\mathcal{D}_{eval} \coloneqq \{\mathbf{T}_i\}_{i=1}^{n}$. Each $\mathbf{T}_i$ is an m-shot learning task associated with its own input space $\mathcal{X}$ and output space $\mathcal{Y}$. As such, $\mathbf{T}_i$ is comprised of $m+1$ tuples of paired input and output arrays, where m tuples are *demonstration pairs* $(\mathbf{x}_{train}, \mathbf{y}_{train}) \in \mathcal{X}^m \times \mathcal{Y}^m$ and one is a *test pair* $(\mathbf{x}_{test}, \mathbf{y}_{test}) \in \mathcal{X} \times \mathcal{Y}$. Thus, $\mathbf{T}_i \coloneqq \{(\mathbf{x}_{train}, \mathbf{y}_{train})\} \cup \{(\mathbf{x}_{test}, \mathbf{y}_{test})\}$. When sufficiently clear, we use $(\mathbf{x}, \mathbf{y})$ to denote a single random input-output pair. Each $\mathbf{T}_i$ is governed by a deterministic *rule* or *transformation* $\delta_i : \mathcal{X} \to \mathcal{Y}$, which maps each input array to its respective output. Let $\mathcal{A}$ be our test-taker (e.g., an LM). To succeed on $\mathbf{T}_i$, $\mathcal{A}$ must learn δ_i sufficiently well to correctly predict $\mathbf{y}_{test}$ from $\mathbf{x}_{test}$ (with predictions denoted as $\widehat{\mathbf{y}}_{test}$). $\mathcal{D}_{eval}$ is presented to $\mathcal{A}$ as string representations of Python arrays. As in Akyürek et al. [1], we encode colors as integers: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9].

3 CausalARC: Abstract Reasoning with Causal World Models

3.1 ARC Through a Causal Lens

To lay the groundwork for CausalARC, we first outline a causal interpretation of the original ARC dataset and introduce the assumptions underlying our design choices.

SCMs: Causal World Models, Causal Programs As in ARC, CausalARC features high task diversity: each task instance is sampled from a unique world model. In program synthesis approaches to ARC, world models are specified as programs [23]. The present work assumes a *causal world model* describing the data generating process, which we define in the formal language of SCMs. As the PCH is well-defined given a fully specified SCM [2], this SCM offers a highly information-rich world model upon which we can perform symbolic operations. Classically, we can express SCMs in mathematical notation (e.g., Example 3.1). Alternatively, we can frame SCMs as *causal programs* that are implemented in a programming language. Under this framing, the act of sampling corresponds to executing the program with different random seeds, intervening corresponds to principled alterations of the program's logic, and counterfactuals correspond to performing multiple interventions on the same random seed (i.e., the exogenous context is held constant, as required by Definition 2.4). The choice to model reasoning tasks as generative causal world models borrows intuition from prior work in reinforcement learning (RL), where causal world models are expressed as Bayesian networks or SCMs [34]. Formally, Richens and Everitt [34] prove that "any agent capable of adapting to a sufficiently large set of distributional shifts must have learned a causal model of the data generating process." Though we do not explore RL, we follow the intuition that learning new causal worlds model on the fly facilitates generalization under distribution shift.

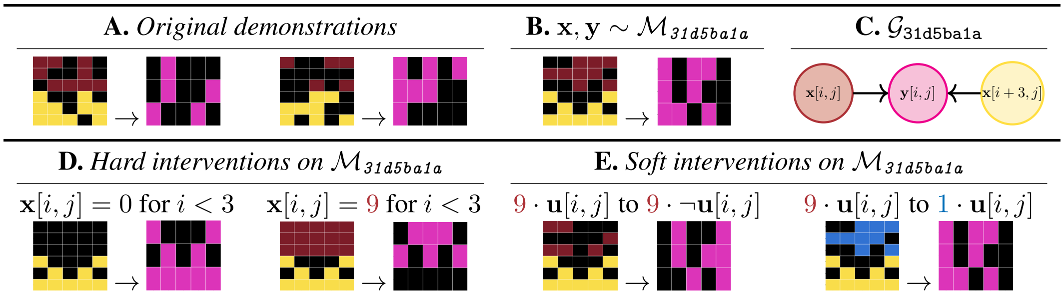

Figure 4: Input-output arrays for ARC-AGI-1 task `31d5ba1a` [5]. (**A**) A subset of the official demonstration pairs $(\mathbf{x}_{train}, \mathbf{y}_{train})$. (**B**) A random sample from SCM $\mathcal{M}_{\texttt{31d5ba1a}}$ defined in Example 3.1. (**C**) Causal DAG $\mathcal{G}_{\texttt{31d5ba1a}}$ representing $\mathcal{M}_{\texttt{31d5ba1a}}$, where $\mathbf{y}[i,j] = 6 \cdot \mathrm{xor}\,(\mathbf{x}[i,j],\ \mathbf{x}[i+3,j])$ for $i \in [0,2], j \in [0,4]$. (**D-E**) Samples from interventional submodels of $\mathcal{M}_{\texttt{31d5ba1a}}$, where exogenous variables are held constant and the causal effects of interventions propagate to $\mathbf{y}$.

Representing Random Variables At the highest level of abstraction, it is clear that $\mathbf{x}$ causes $\mathbf{y}$. We can denote this graphically as $(\mathbf{x}) \rightarrow (\mathbf{y})$, where $(\mathbf{x})$ and $(\mathbf{y})$ are multivariate "supernodes" in the causal DAG. At finer granularity, we could choose to model each exogenous variable as Bernoulli (Example 3.1) or as a categorical random variable that can take on values 0 through 9 (Example C.1). Arbitrary causal functions could result in endogenous random variables with complex distributions. Additionally, an array can be described by grid-level features over all or some elements (e.g., the total number of pixels taking a specific value, the total number of specific sprites, etc.). These can also be modeled as random variables, as in Example C.1. Note that all stochasticity in our model arises from the random sampling of our exogenous variables, and we restrict our attention to deterministic causal functions for endogenous variables. Any $\mathbf{x}$ maps to only one $\mathbf{y}$, just as in the original ARC.

Grids as Samples from Latent SCMs We assume that an underlying SCM $\mathcal{M}_i$ exists for each $\mathbf{T}_i$. Thus, we assume that every array in $\mathbf{T}_i$ is sampled from the SCM (denoted $\mathbf{T}_i \sim \mathcal{M}_i$). In some original ARC tasks, a unique $\mathcal{M}_i$ can be easily derived from the limited examples provided by $(\mathbf{x}_{train}, \mathbf{y}_{train})$. In other cases, multiple SCMs might be compatible with the limited information provided by $(\mathbf{x}_{train}, \mathbf{y}_{train})$, yielding some form of equivalence class. This latter case is analogous to the causal discovery setting where the full unique DAG is not identifiable, though its Markov equivalence class is [39]. To support the assumption that $\mathbf{T}_i \sim \mathcal{M}_i$, we consider cases where an SCM is easily recoverable from observing $(\mathbf{x}_{train}, \mathbf{y}_{train})$ (Examples 3.1, C.1). Samples from the recovered SCMs are indistinguishable from those provided in the official task (Figures 4, C.1).

Example 3.1 (A fully recovered SCM). *Consider ARC-AGI-1 task `31d5ba1a` (expert level [21, 22]; Figure 4). We can define an SCM $\mathcal{M}_{\texttt{31d5ba1a}} = \langle \mathbf{U}, p(\mathbf{u}), \mathbf{V}, \mathcal{F} \rangle$ where $\mathbf{V}$ are the elements of our arrays, $f \in \mathcal{F}$ are logical* xor[4] *and/or scalar multiplication, and $p(\mathbf{u})$ is Bernoulli.*

$$\mathbf{u}[i,j] \sim \mathrm{Ber}(0.5) \qquad \textit{for } i \in [0,5], j \in [0,4] \tag{1}$$

$$\mathbf{x}[i,j] = \begin{cases} 9 \cdot \mathbf{u}[i,j] & \textit{if } i < 3 \\ 4 \cdot \mathbf{u}[i,j] & \textit{else} \end{cases} \qquad \textit{for } i \in [0,5], j \in [0,4] \tag{2}$$

$$\mathbf{y}[i,j] = 6 \cdot \mathrm{xor}\,(\mathbf{x}[i,j],\ \mathbf{x}[i+3,j]) \qquad \textit{for } i \in [0,2], j \in [0,4]. \tag{3}$$

With our formally defined SCM, we can perform hard or soft interventions and observe the impacts on output $\mathbf{y}$. When we hold $\mathbf{U} = \mathbf{u}$ constant, we claim the resulting arrays are *counterfactuals* (Definition 2.4; Figure 5). As shown in Figure 4D, hard interventions on $\mathbf{x}$ impact the color distribution over $\mathbf{y}$. In Figure 4E, we observe that soft interventions on the causal function for $\mathbf{x}[i,j]$ may or may not impact the output $\mathbf{y}$. As the color of $\mathbf{y}[i,j]$ is impacted by whether $\mathbf{x}[i,j]$ is zero or nonzero but not by the specific nonzero value taken, interventions performed in Figure 4E (right) do not influence $\mathbf{y}$.

Rules as Observable Proxies for SCMs In practice, the SCM describing a random ARC task will be too complex to easily derive by hand from limited examples. Thus, we proceed under the assumption that the true SCM $\mathcal{M}_i$ explaining $\mathbf{T}_i$ is unknown. We can choose to model rule δ_i as an

[4] Our notation follows the `numpy` implementation of logical operators, where any nonzero value is treated as the Boolean 1 = TRUE while zero is treated as FALSE. All logical operators return integer values in $[0, 1]$.

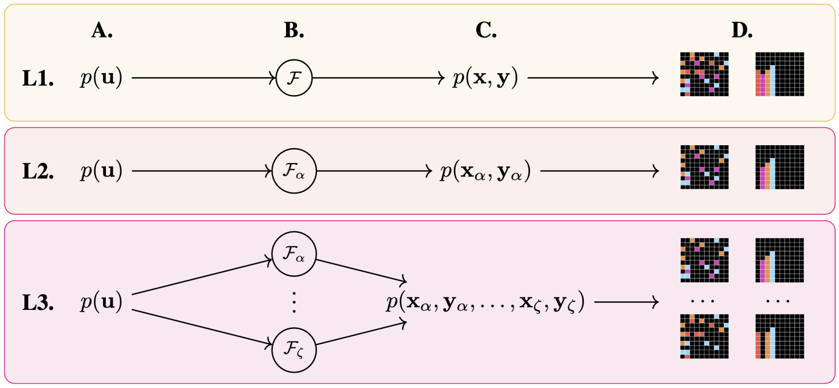

Figure 5: Jointly observed counterfactuals in CausalARC. L1, L2, and L3 denote the rungs of the PCH (Figure 1). (**A**) The distribution over the exogenous context (i.e., the external state). (**B**) Transformations applied to the exogenous context (e.g., functions $\mathcal{F}$ in the observational world; updated functions $\mathcal{F}_\alpha$ under intervention α). (**C**) Induced distributions, following from the applied transformation. (**D**) CausalARC samples from each rung of the PCH. Adapted from [2] (Figure 27.2).

observable and human-interpretable proxy for the latent $\mathcal{M}_i$. In practice, however, δ_i is likely to be a lossy distillation of $\mathcal{M}_i$: often, $\mathcal{A}$ (human or machine) only learns as much about $\mathcal{M}_i$ as is needed to output $\widehat{\mathbf{y}}_{test}$ (for example, parameter p to the Bernoulli distribution over the exogenous variables is not generally needed to correctly estimate $\widehat{\mathbf{y}}_{test}$). However, without access to the data generating process provided by $\mathcal{M}_i$, we cannot randomly sample from the true distribution nor guarantee that data augmentations yield true counterfactuals. We introduce CausalARC to address these limitations.

3.2 Constructing CausalARC

Task Generation All tasks were constructed in Python using built-in functions and `numpy`. The task construction procedure was as follows. First, an SCM was defined in mathematical notation. This SCM was then manually translated to Python, with functionality for performing hard and/or soft interventions and obtaining sample distributions. Task instances sampled from SCMs are returned as dictionaries in a format consistent with the official ARC dataset[5] to ensure compatibility with existing ARC pipelines. To date, the static CausalARC dataset is composed of 50 task instances, each with five demonstration pairs, one test pair, and 5–10 counterfactual pairs per demonstration pair.[6]

Causal Annotations In addition to counterfactual examples, each task dictionary is annotated by a human expert with at least one representation of the causal world model: (1) in all cases, a string representation of the Python program is included; (2) in a subset of tasks, the mathematical notation for the SCM is provided; and (3) in another (non-disjoint) subset, the adjacency matrix of the causal graph is provided. Representation (1) is potentially useful for program synthesis settings (e.g., for supervising induction models, as in [23]), while (3) is especially useful for causal discovery use cases.

Task Themes To date, tasks in CausalARC fall into the following human-labeled categories:

1. *Counting.* Tasks require elements to be counted (e.g., total pixels of a certain color).
2. *Extension.* Paths or sprites must be extended according to some rule.
3. *Logical.* Tasks require reasoning over causal functions that are logical operators.
4. *Ordering.* Elements must be ordered according to some rule, colored according to some order, etc.

In the static dataset, each theme currently features 10 task instances (with the exception of logical tasks, for which there are 20). As with the concept labels in ConceptARC [29] and the difficulty labels provided by LeGris et al. [21, 22], task labels allow the user to do fine-grained error analyses. For example, one could compare the performance of test-taker $\mathcal{A}$ on logical tasks when causal functions are *xor* versus *and*, or tasks requiring counting versus ordering, etc.

[5] https://www.kaggle.com/competitions/arc-prize-2024

[6] Hugging Face link at camera-ready.

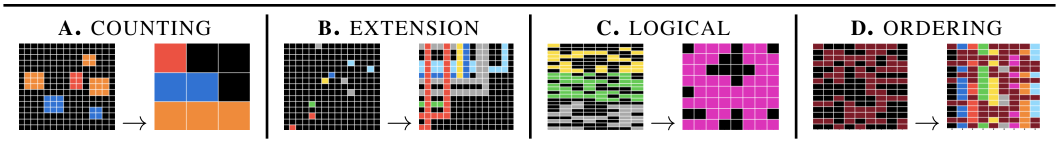

Figure 6: Example demonstration pairs for CausalARC themes.

Jointly Observed Counterfactuals The fundamental problem of causal inference is that counterfactuals cannot be jointly observed in the physical universe. In our synthetic grid world, we can exploit the fact that we have full control over the data generating process and exogenous factors can be held constant across multiple interventions. Thus, this synthetic environment allows for the juxtaposition of 'treated' samples under multiple interventions alongside their 'untreated' counterparts, without violating the assumption that $\mathbf{U} = \mathbf{u}$ in all instances. In this way, we justify our choice to model the data augmentations on $\mathbf{T}_i$ as a set of jointly observed counterfactuals (Figure 5).

Prompt Sampling CausalARC provides functionality for randomly sampling prompts from the underlying SCM according to user specifications. Currently, the following aspects can vary: total number of in-context demonstrations, type of in-context demonstrations (all L1 versus alternating L1/L3), and query theme (counterfactual reasoning, abstract reasoning, program synthesis, and causal discovery). For counterfactual reasoning, the LM is prompted to predict the output under intervention for a previously seen L1 example (e.g., Figures D.4, D.5). For abstract reasoning, the LM is prompted to predict the output array for an L1 input, as in conventional ARC setups [1]. For program synthesis, the LM is prompted to generate a Python program that expresses the SCM (e.g., Figures D.6, D.7). For causal discovery, the LM is prompted to predict a property of the underlying SCM, such as the causal parents of an array element or the form of a causal function (e.g., Figures D.8, D.9).

4 Empirical Demonstrations

As a proof-of-concept, we illustrate four potential use cases for CausalARC: (1) abstract reasoning with TTT, (2) counterfactual reasoning with in-context learning, (3) program synthesis, and (4) causal discovery with logical reasoning. Full experimental details and results are in Appendix D, including a full description of the tasks used for each experiment.

Models Models were selected for diversity. We compare designated reasoning models (e.g., o4-mini) versus vanilla LMs (e.g., GPT-4o mini); older, less powerful models (e.g., Claude Haiku 3.5) versus newer, more powerful models (e.g., Claude Sonnet 4); and open-source models (e.g., LLama 3 8B, Llama 4 Scout 17B) versus closed-source models (all others). Full model details are in Table D.1.

Prompt Formulation We explored impacts of prompt formulation with respect to PCH level and total number of in-context examples. To isolate the effects of PCH level on LM performance, we compared prompts with L1 versus alternating L1/L3 in-context demonstrations while holding the total number of demonstrations constant (e.g., Figure D.6 versus D.7).

Metrics As in Akyürek et al. [1], LM performance was measured by output accuracy (i.e., whether the output array was exactly correct) and relative Hamming distance (HD; i.e., the number of positions where the true array differs from the predicted array, normalized by the number of elements in the true array). For HD, a score of 0 indicates correctness while a score of 1 indicates complete failure (including cases where the LM returns a malformed array or no response at all).

4.1 Abstract Reasoning with Test-Time Training

Motivation To gauge the difficulty of CausalARC, we benchmarked MARC with TTT [1] on the full static dataset. MARC was the second-place paper winner for ARC-AGI-1. It takes a neural transduction approach using a Llama 3 8B base model fine-tuned on large ARC-like datasets, with TTT plus in-context learning at inference. Automated data augmentations (e.g., geometric transformations and color permutations) increase the sample size of the TTT dataset for each task. We employ MARC as it is provided in the public code base, with no modifications.[7]

[7] https://github.com/ekinakyurek/marc

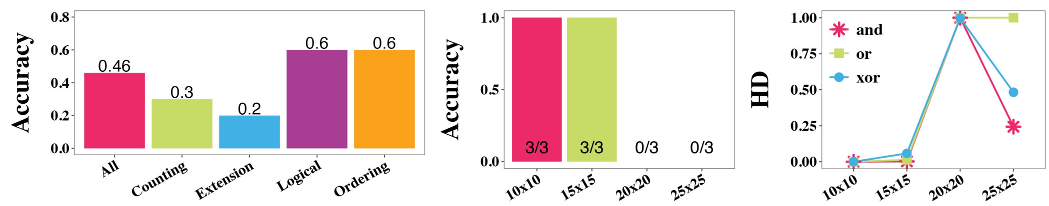

Figure 7: (Left) Accuracy by CausalARC theme for MARC with TTT (Llama 3 8B base) [1]. (Center and right) Performance on *and*, *or*, and *xor* tasks sampled from `SCMdky5` as array size increases.

Results & Discussion Overall accuracy on CausalARC was 46%, with significant variation across themes (Figure 7, left). This is similar to MARC's pure transduction score of 47.1% accuracy on ARC-AGI-1 [1], suggesting that CausalARC is of comparable difficulty. Extension and counting were significantly more challenging for MARC than logic and ordering, though we cannot establish that this is due to the underlying concepts versus other artifacts of these task distributions. MARC shows signs of struggling with large output arrays on both ARC-AGI-1 and CausalARC, as exemplified by steep dropoffs in performance for logical tasks as array size increases (Figure 7, center).

4.2 Counterfactual Reasoning with In-Context Learning

Motivation This experiment demonstrates the use of CausalARC for counterfactual reasoning evaluation with few-shot, in-context learning demonstrations. For each task, three in-context demonstrations were presented before a final test case that prompted the LM to predict the counterfactual output for a previously seen L1 demonstration. In addition to measuring the ability of the LM to predict counterfactual samples from the underlying world model, this experiment compares performance when in-context demonstrations are sampled from L1 versus L3.

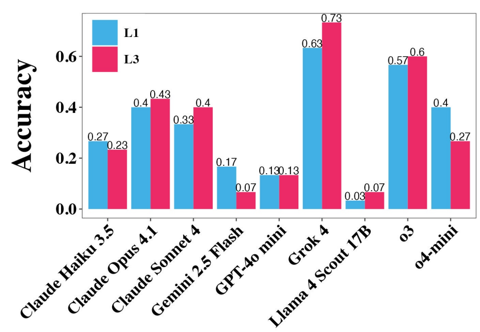

Figure 8: Counterfactual reasoning on six counting, extension, and ordering tasks. Scores are over five random prompt samples.

Results & Discussion Performance varied widely across models (Figures 8, D.13, D.14, D.15). L3 demonstrations did not consistently confer benefits. This result could arise from lower sample diversity in L3 prompts (as L3 demonstrations are often similar to their L1 counterpart, due to the exogenous context being fixed). Strong performance from recent closed-source models on logical tasks (Figure D.14) could be an artifact of fine-tuning on ARC, as these tasks are similar to ARC-AGI-1 task `31d5ba1a` (which used *xor* functions; Example 3.1).

4.3 Program Synthesis

Motivation Winning ARC strategies fall into two dominant camps [5]: (1) neural transduction approaches, where $\widehat{\mathbf{y}}_{test}$ is directly predicted (as in MARC); and (2) neural induction with program synthesis, where the neural network outputs a program implementing δ_i that is executed to obtain $\widehat{\mathbf{y}}_{test}$. As the first-place ARC-AGI-1 paper winner, BARC [23] took a meta-learning approach [28] that combined neural transduction with Python program synthesis, demonstrating improved performance relative to transduction or induction alone. MARC saw significant performance gains when ensembling with BARC, further supporting the combined use of induction and transduction [1]. Given the success and community interest surrounding program synthesis for reasoning, we also apply CausalARC for this use case. Accuracy was measured with respect to the output $\widehat{\mathbf{y}}_{test}$ generated by passing a test input $\mathbf{x}_{test}$ to the LM-generated program.

Results & Discussion Total in-context examples had a notable impact on program correctness (Figures 9, D.16). Maximum performance was attained by o4-mini, which showed possible signs of fine-tuning for ARC-like program synthesis (Figure D.10). Accuracy peaked at six in-context examples for all models except Sonnet 4 (which saw monotonic increases) and Llama 4 Scout 17B (which failed to output any correct programs). L3 prompts rarely conferred benefits (Figure D.16).

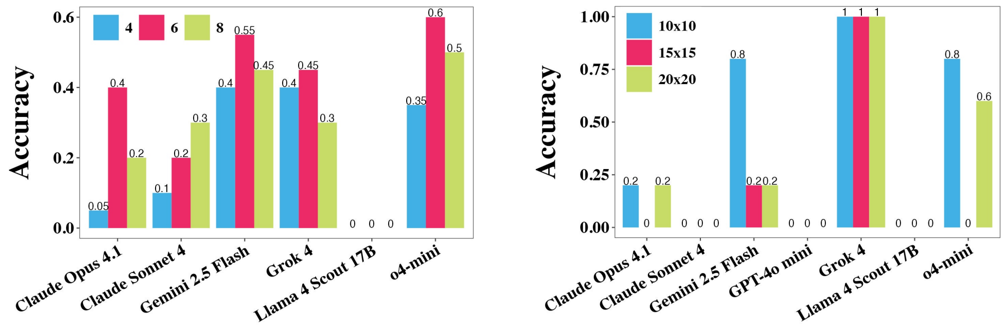

Figure 9: (Left) Program synthesis on four counting and extension tasks as total in-context demonstrations increased. (Right) Causal discovery with logical reasoning as array size increased. Scores were over five random L1 prompt samples. Results for L3 prompts are in Appendix D.2

4.4 Causal Discovery with Logical Reasoning

Motivation Causal discovery is the body of theory and methods concerned with learning causal graphical structures from data [8]. Though there is increasing interest in using LMs to support causal discovery, recent evaluations show steep shortcomings [42, 18, 6]. As CausalARC tasks are annotated with fully specified SCMs and (in a subset) the corresponding adjacency matrix, CausalARC is conducive to benchmarking LMs on causal discovery tasks. As a proof-of-concept, we tested the performance of state-of-the-art LMs on causal discovery with logical reasoning with respect to scaling problem complexity (array size). Rather than directly asking for adjacencies in the causal graph, LMs were asked to predict the logical operators acting as causal functions (which implies knowledge of the causal parents of each output cell). Example prompts are provided in Figures D.8 and D.9.

Results & Discussion Performance varied widely, with Grok 4 consistently outperforming (Figures 9, D.17). All other models displayed unstable performance, as monotonic declines in accuracy were not consistently observed. L3 prompts did not consistently confer benefits both within and across models, though benefits were greater than for program synthesis (Figure D.17). Correct responses varied in the extent to which they explicitly enumerated parent-child relationships (Figures D.11, D.12). Future work could query LMs directly for adjacencies, using causal discovery algorithms as baselines. As a proof-of-concept, we ran the popular constraint-based PC algorithm [39] on L1 distributions sampled from `SCMdky5` (Table D.4, Figure D.18). Poor performance of PC on samples with *xor* causal function is likely due to a violation of faithfulness: the common assumption that statistical independencies correspond to separations in the underlying graph. Logical *xor* and deterministic functions both famously cause faithfulness violations, which can significantly undermine causal discovery [26]. This may suggest that CausalARC could be extended for testing robustness to faithfulness violations in discrete data, a problem previously explored for continuous data [31].

5 Conclusion

Limitations & Future Directions We introduce CausalARC as an open-ended AI reasoning testbed that accommodates diverse experimental setups, including TTT and in-context learning for counterfactual, abstract, and logical reasoning. Results presented in this proof-of-concept are preliminary and require further investigation. While the manual design of this dataset allows us to maintain strict causal assumptions, this also inhibits scaling and prevents use for fine-tuning and large-scale benchmarking. Future work could expand CausalARC with additional tasks or task themes.

Though within- and between-model performance varied heavily across tasks, there were some signs that state-of-the-art proprietary models benefited from fine-tuning on ARC and its extensions (e.g., o4-mini offered "ARC-style input/output" in some program synthesis responses, despite ARC never being mentioned in the prompt; Claude Sonnet 4 volunteered to perform program synthesis on counterfactual reasoning tasks without being prompted to do so; and Grok 4's exceptional performance on *xor* logical problems, which are known to exist in ARC-AGI-1). Despite this, we do not observe consistent evidence of saturation on CausalARC in state-of-the-art models.

References

[1] E. Akyürek, M. Damani, A. Zweiger, L. Qiu, H. Guo, J. Pari, Y. Kim, and J. Andreas. The surprising effectiveness of test-time training for few-shot learning. *International Conference on Machine Learning*, 2025.

[2] E. Bareinboim, J. D. Correa, D. Ibeling, and T. Icard. *On Pearl's Hierarchy and the Foundations of Causal Inference*, page 507–556. Association for Computing Machinery, New York, NY, USA, 1 edition, 2022. ISBN 9781450395861. URL https://doi.org/10.1145/3501714.3501743.

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 2020.

[4] F. Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

[5] F. Chollet, M. Knoop, G. Kamradt, and B. Landers. Arc prize 2024: Technical report. *arXiv preprint arXiv:2412.04604*, 2024.

[6] J. Gao, X. Ding, B. Qin, and T. Liu. Is chatgpt a good causal reasoner? a comprehensive evaluation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 11111–11126, 2023.

[7] H. Geffner, R. Dechter, and J. Y. Halpern. *Probabilistic and Causal Inference: The Works of Judea Pearl*. ACM, 2022.

[8] C. Glymour, K. Zhang, and P. Spirtes. Review of causal discovery methods based on graphical models. *Frontiers in genetics*, 10:524, 2019.

[9] M. K. Goddu and A. Gopnik. The development of human causal learning and reasoning. *Nature Reviews Psychology*, pages 1–21, 2024.

[10] A. Gong, K. Stankevičiūtė, C. Wan, A. Kabra, R. Thesmar, J. Lee, J. Klenke, C. P. Gomes, and K. Q. Weinberger. Phantomwiki: On-demand datasets for reasoning and retrieval evaluation. In *International Conference on Machine Learning*, 2025.

[11] J. González and A. Nori. Does reasoning emerge? examining the probabilities of causation in large language models. *Advances in Neural Information Processing Systems*, 37:117737–117761, 2024.

[12] D. Ha and J. Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[13] S. Hao, Y. Gu, H. Ma, J. Hong, Z. Wang, D. Wang, and Z. Hu. Reasoning with language model is planning with world model. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Singapore, Dec. 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.507. URL https://aclanthology.org/2023.emnlp-main.507/.

[14] M. Hodel. Addressing the abstraction and reasoning corpus via procedural example generation, 2024. URL https://arxiv.org/abs/2404.07353.

[15] Q. Huang. Model-based or model-free, a review of approaches in reinforcement learning. In *2020 International Conference on Computing and Data Science (CDS)*, pages 219–221. IEEE, 2020.

[16] A. Hüyük, X. Xu, J. Maasch, A. V. Nori, and J. González. Reasoning elicitation in language models via counterfactual feedback. *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.03767.

[17] Z. Jin, Y. Chen, F. Leeb, L. Gresele, O. Kamal, Z. Lyu, K. Blin, F. Gonzalez Adauto, M. Kleiman-Weiner, M. Sachan, et al. Cladder: Assessing causal reasoning in language models. *Advances in Neural Information Processing Systems*, 2023.

[18] Z. Jin, J. Liu, Z. Lyu, S. Poff, M. Sachan, R. Mihalcea, M. Diab, and B. Schölkopf. Can large language models infer causation from correlation? *arXiv preprint arXiv:2306.05836*, 2023.

[19] B. M. Lake and M. Baroni. Human-like systematic generalization through a meta-learning neural network. *Nature*, 623(7985):115–121, 2023.

[20] B. M. Lake, T. Linzen, and M. Baroni. Human few-shot learning of compositional instructions. *arXiv preprint arXiv:1901.04587*, 2019.

[21] S. LeGris, W. K. Vong, B. M. Lake, and T. M. Gureckis. H-arc: A robust estimate of human performance on the abstraction and reasoning corpus benchmark. *arXiv preprint arXiv:2409.01374*, 2024.

[22] S. LeGris, W. K. Vong, B. M. Lake, and T. M. Gureckis. A comprehensive behavioral dataset for the abstraction and reasoning corpus. *Scientific Data*, 2025. doi: https://doi.org/10.1038/s41597-025-05687-1.

[23] W.-D. Li, K. Hu, C. Larsen, Y. Wu, S. Alford, C. Woo, S. M. Dunn, H. Tang, M. Naim, D. Nguyen, et al. Combining induction and transduction for abstract reasoning. *International Conference on Learning Representations*, 2025.

[24] J. Liang, R. He, and T. Tan. A comprehensive survey on test-time adaptation under distribution shifts. *International Journal of Computer Vision*, 133(1):31–64, 2025.

[25] J. Maasch, A. Hüyük, X. Xu, A. V. Nori, and J. Gonzalez. Compositional causal reasoning evaluation in language models. *International Conference on Machine Learning*, 2025.

[26] A. Marx, A. Gretton, and J. M. Mooij. A weaker faithfulness assumption based on triple interactions. In *Uncertainty in Artificial Intelligence*, 2021.

[27] S. I. Mirzadeh, K. Alizadeh, H. Shahrokhi, O. Tuzel, S. Bengio, and M. Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *International Conference on Learning Representations*, 2025.

[28] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel. A simple neural attentive meta-learner. In *International Conference on Learning Representations*, 2018.

[29] A. Moskvichev, V. V. Odouard, and M. Mitchell. The conceptarc benchmark: Evaluating understanding and generalization in the arc domain. *Transactions on Machine Learning Research*, 2023.

[30] S. Niu, J. Wu, Y. Zhang, Y. Chen, S. Zheng, P. Zhao, and M. Tan. Efficient test-time model adaptation without forgetting. In *International Conference on Machine Learning*. PMLR, 2022.

[31] M. Olko, M. Gajewski, J. Wojciechowska, M. Morzy, P. Sankowski, and P. Miłoś. Since faithfulness fails: The performance limits of neural causal discovery. In *International Conference on Machine Learning*, 2025.

[32] J. Pearl. Causality: Models, reasoning, and inference. *Cambridge, UK: Cambridge University Press*, 19(2):3, 2000.

[33] J. Pearl. Structural counterfactuals: A brief introduction. *Cognitive Science*, 37(6):977–985, 2013.

[34] J. Richens and T. Everitt. Robust agents learn causal world models. *International Conference on Learning Representations*, 2024.

[35] J. Richens, T. Everitt, and D. Abel. General agents need world models. In *International Conference on Machine Learning*, 2025.

[36] P. Shojaee, I. Mirzadeh, K. Alizadeh, M. Horton, S. Bengio, and M. Farajtabar. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. *arXiv preprint arXiv:2506.06941*, 2025.

[37] R. B. Shrestha, S. Malberg, and G. Groh. From causal parrots to causal prophets? towards sound causal reasoning with large language models. In *Proceedings of the 5th International Conference on Natural Language Processing for Digital Humanities*, pages 319–333, 2025.

[38] E. S. Spelke and K. D. Kinzler. Core knowledge. *Developmental science*, 10(1):89–96, 2007.

[39] P. Spirtes, C. N. Glymour, and R. Scheines. *Causation, Prediction, and Search*. MIT press, 2000.

[40] Y. Sun, X. Wang, Z. Liu, J. Miller, A. Efros, and M. Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *International Conference on Machine Learning*, pages 9229–9248. PMLR, 2020.

[41] J. Tian and J. Pearl. Probabilities of causation: Bounds and identification. *Annals of Mathematics and Artificial Intelligence*, 28(1):287–313, 2000.

[42] R. Tu, C. Ma, and C. Zhang. Causal-discovery performance of chatgpt in the context of neuropathic pain diagnosis. *arXiv preprint arXiv:2301.13819*, 2023.

[43] J. Wei, J. Wei, Y. Tay, D. Tran, A. Webson, Y. Lu, X. Chen, H. Liu, D. Huang, D. Zhou, et al. Larger language models do in-context learning differently. *arXiv preprint arXiv:2303.03846*, 2023.

[44] X. Xu, R. Lawrence, K. Dubey, A. Pandey, R. Ueno, F. Falck, A. V. Nori, R. Sharma, A. Sharma, and J. Gonzalez. Re-imagine: Symbolic benchmark synthesis for reasoning evaluation. In *International Conference on Machine Learning*, 2025.

[45] M. Zečević, M. Willig, D. S. Dhami, and K. Kersting. Causal parrots: Large language models may talk causality but are not causal. *Transactions on Machine Learning Research*, 2024.

Appendix

Table of Contents

A Impact Statement

The possibility of AI reasoning emergence has broad scientific, economic, and social implications, including matters of safety and fairness. Though this work aims to promote rigorous reasoning evaluation, strong performance on CausalARC is *necessary but not sufficient* for demonstrating that LMs can reason. Results should be interpreted with great caution when deploying LMs for reasoning tasks that could have safety or fairness consequences.

B Extended Preliminaries

B.1 Defining and Measuring Reasoning

Any framework claiming to measure reasoning should clearly define its target of measure. Reasoning is a famously difficult concept to concretely define and has taken on many meanings across time and research domains. To begin, we first declare our chosen definitions of *generalization* and *intelligence*. We take a definition of generalization that requires adaptability in novel settings.

Definition B.1 (Generalization, Chollet [4])**.** The ability to handle scenarios or tasks that differ from previously encountered situations, demonstrating both *robustness* and *flexibility*.

Definition B.2 (Robustness, Chollet [4])**.** Adaptation to *known unknowns* within a single task or well-defined set of tasks.

Definition B.3 (Flexibility, Chollet [4])**.** Adaptation to *unknown unknowns* across a broad category of related tasks.

In CausalARC, an example of robustness would be high accuracy on a large number of test cases sampled from one causal world model on which the test-taker has undergone TTT. An example of flexibility would be high accuracy on a large number of unique causal world models.

We follow Chollet [4] in defining the intelligence of a system as a measure of *skill-acquisition efficiency*, rather than skill itself. This is consistent with the conventional wisdom that human intelligence tests should measure cognitive capacities in a general sense, and not task-specific skills acquired through experience or practice [4].

Definition B.4 (Intelligence, Chollet [4])**.** Skill-acquisition efficiency over a range of tasks, controlling for priors, experience, and generalization difficulty.

Thus, we take a *general intelligence* to be one with high skill acquisition efficiency and high generalization over a large range of tasks. We highlight the need to control for experience and priors, as “unlimited priors or experience can produce systems with little-to-no generalization power (or intelligence) that exhibit high skill at any number of tasks” [4]. This latter case (where skill outpaces generalization) is reminiscent of current shortcomings in large LM reasoning [27, 36, 44], where pretraining on web-scale corpora and extensive fine-tuning could be seen as nearly “ulimited priors or experience.”

Finally, we define our notions of *reasoning*: cognitive processes that are core attributes of intelligent thinking. We focus on abstract, logical, and counterfactual reasoning, as these are the forms tested in the current iteration of CausalARC.

Definition B.5 (Abstract reasoning)**.** The process of drawing valid conclusions about novel visual information by identifying patterns using innate cognitive priors, rather than the accumulation of concrete, domain-specific knowledge.

Here, validity is measured by correctness (e.g., accuracy). Innate cognitive priors are primitives acquired at birth or early development under minimal supervision, broadly pertaining to concepts like objects, actions, number, and space (e.g., a grasp of intuitive physics, such as gravity; basic numeracy; basic geometric operations, such as rotation; a sense of relative size or magnitude, etc.) [38].

Definition B.6 (Logical reasoning)**.** The process of drawing valid conclusions by applying formal logical rules to novel information.

Logical reasoning encompasses many subtypes, including deductive, inductive, and abductive. In this work, we primarily focus on the process of applying logical operators in Boolean algebra.

Next, recall the definition of a counterfactual: *the counterfactual Y_x under model $\mathcal{M}$ is given by* $Y_x(\mathbf{u}) \coloneqq Y_{\mathcal{M}_x}(\mathbf{u})$ (Definition 2.4). We define counterfactual reasoning as follows.

Definition B.7 (Counterfactual reasoning)**.** The process of constructing valid answers to “what if” questions about imagined alternate outcomes in novel settings, as in: “What value would Y have taken had X been x in context $\mathbf{U} = \mathbf{u}$?”

Then, the *ability to reason* abstractly, logically, or counterfactually is the ability to correctly execute these reasoning processes with robust and flexible generalization. Benchmarks like ARC and CausalARC are designed to measure signs of this ability by presenting the test-taker with diverse and novel problems while controlling for experience and priors. Nevertheless, these benchmarks measure the end-point of reasoning: i.e., the extent to which the final output is correct. Under Definitions B.5, B.6, and B.7, reasoning is a *process* and not an end-product. Thus, there is increasing interest in benchmarks that also assess the intermediate steps of AI reasoning [36]. Additionally, fairly controlling for experience and priors is challenging in the context of large LMs that do not release details of their pretraining and fine-tuning, and it is known that state-of-the-art LMs generally have prior knowledge of ARC. These are just some reasons that we argue that strong performance on CausalARC and analogous benchmarks is *necessary but not sufficient* for demonstrating that LMs can reason.

B.2 World Models

Emergent behaviors in large LMs have raised new questions on whether learning accurate world models can improve AI reasoning [13]. Notably, world models have been variously defined across literatures [12]. In model-based reinforcement learning (RL), a world model is often a predictor of the evolution of an environment under arbitrary policies (e.g., an approximation of the transition function of a Markov decision process) [34]. While the usefulness of world models in RL has been a matter of significant debate [15], the necessity of world models has been proven for multiple settings. Given a sufficiently diverse set of *goal-directed tasks*, any agent satisfying a regret bound for these tasks will necessarily have learned a *predictive world model* that converges to the true model as the agent approaches optimality [35]. Additionally, given regret-bounded policies for a sufficiently large set of *distributional shifts*, the agent necessarily learns an approximate *causal world model* that converges to the true causal model under optimal policies [34]. Richens and Everitt [34] express causal world models as causal Bayesian networks (DAGs), though their results extend to fully specified SCMs. While the present work does not explore CausalARC in the context of RL, we are motivated by the intuition provided by [34, 35] that world models can facilitate (and can even be necessary for) optimal decision-making, planning, and reasoning in AI.

C ARC Through a Causal Lens: Extended Discussion

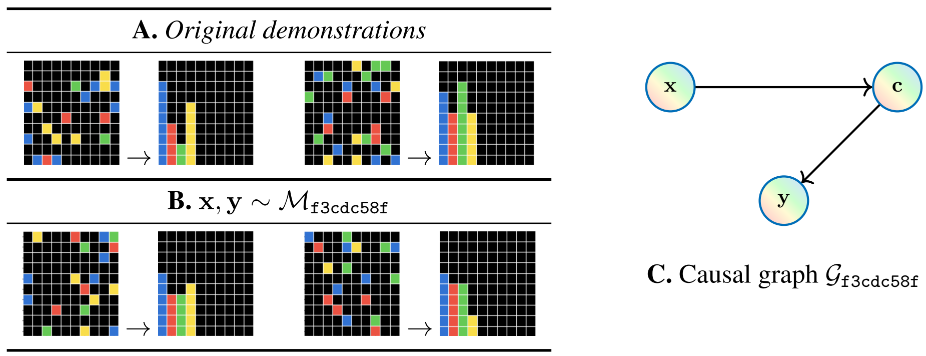

Figure C.1: Input-output arrays for ARC-AGI-1 task `f3cdc58f`. (**A**) A subset of official demonstration pairs $(\mathbf{x}_{train}, \mathbf{y}_{train})$. (**B**) Randomly sampled pairs from the SCM defined in Example C.1. (**C**) A DAG representation for task `f3cdc58f`, where $\mathbf{c}$ is an array-level feature over $\mathbf{x}$.

We present another example of an ARC-AGI-1 task where the SCM is easily recoverable by observing $(\mathbf{x}_{train}, \mathbf{y}_{train})$. In this case, the SCM includes an endogenous variable that is an array-level feature (count of total pixels per color). Note that when indexing in arrays, we use notation similar to Python (e.g., $\mathbf{y}[:, -1]$ denotes all rows of $\mathbf{y}$ for the final column).

Example C.1 (A fully recovered SCM with array-level features)**.** *Consider ARC-AGI-1 task* `f3cdc58f` *(easy level [21, 22]; Figure C.1). We can define an SCM* $\mathcal{M}_{\texttt{f3cdc58f}} = \langle \mathbf{U}, p(\mathbf{u}), \mathbf{V}, \mathcal{F} \rangle$ *where* $\mathbf{V}$ *are the elements of our arrays,* $f \in \mathcal{F}$ *are expressed in Equations 5–7,* $p(\mathbf{u})$ *is categorical with support* $\mathcal{X}$*, and endogenous variables in output array* $\mathbf{y}$ *are causal children of an array-level feature over* $\mathbf{x}$ *(the total number of elements in* $\mathbf{x}$ *that are a certain color).*

$$\mathbf{u} \sim \text{Cat}(\text{len}(\mathcal{X}), p) \quad \textit{for } \mathcal{X} = [0, 1, 2, 3, 4], \; p = [0.8, 0.05, 0.05, 0.05, 0.05] \tag{4}$$

$$\mathbf{x} = \text{id}_{\mathbf{u}} \tag{5}$$

$$\mathbf{c} = [\text{count}(\mathbf{x} == j)]_{j \in \mathcal{X}} \tag{6}$$

$$\mathbf{y}[-\mathbf{c}[j] :, j-1] = \begin{cases} j & \textit{if } j \in \mathcal{X} \setminus \{0\} \\ 0 & \textit{else.} \end{cases} \tag{7}$$

D Extended Empirics

D.1 Experimental Details

Compute Resources All LM experiments were run on an AWS EC2 g6e.xlarge instance featuring one GPU with 48GB GPU memory and four vCPUs.[8] All other experiments were run locally on a MacBook Pro (Apple M2 Pro chip, 16 GB memory).

Models Benchmarking experiments used the models described in Table D.1. All models used default hyperparameters and temperature 0.0 (except o4-mini, which required temperature 1.0). Models were queried using the `langchain` Python package.[9]

Prompt Formulation Several experiments compare the impacts of L1 versus L3 in-context demonstrations on LM performance. To prevent the conflation of benefits from PCH level versus query length, the total number of in-context demonstrations was held constant across L1 and L3 prompts. Headers for L3 demonstrations cause a small increase in prompt length, as these are slightly longer than L1 example headers. For example, the L1 prompt in Figure D.4 is 4130 characters long, while the L3 prompt in Figure D.5 is 4267 characters. In this case, including L3 demonstrations resulted in a 3.2% length increase.

[8] https://aws.amazon.com/ec2/instance-types/g6e/

[9] https://python.langchain.com/

PROVIDER	MODEL	API ID	URL
OpenAI	GPT-4o mini o4-mini	gpt-4o-mini-2024-07-18 o4-mini-2025-04-16	https://platform.openai.com/docs/models
Anthropic	Claude Haiku 3.5 Claude Sonnet 4	claude-3-5-haiku-20241022 claude-sonnet-4-20250514	https://docs.anthropic.com/en/docs/about-claude/models/
Google	Genini 2.5 Flash	gemini-2.5-flash	https://ai.google.dev/gemini-api/docs/models
XAI	Grok 4	grok-4	https://docs.x.ai/docs/models
Meta (Bedrock)	Llama 4 Scout 17B	us.meta.llama4-scout-17b-instruct-v1:0	https://llama.developer.meta.com/docs/models/

Table D.1: Models used in benchmarking experiments.

D.1.1 Counterfactual Reasoning with In-Context Learning

Tasks Counterfactual reasoning experiments used task instances sampled from all task themes. Counting tasks were `SCMfuy3` and `SCMm5ob`. Extending tasks were `SCMz750` and `SCMwoev`. Ordering tasks were `SCMffb8` and `SCMtzlq`. All tasks with tunable array sizes (all except `SCMfuy3` and `SCMffb8`) used size 25×25.

Logical counterfactual reasoning tasks used variants of `SCMdky5`, `SCMu3am`, and `SCMtcbq`, which are based to varying degrees on ARC-AGI-1 task `31d5ba1a` (Example 3.1). Causal functions for `SCMdky5` were a single logical operator (*and*, *or*, *xor*) while `SCMu3am` and `SCMtcbq` used two logical operators. Task `SCMdky5` was generated for each logical operator at three dimensionalities (10×10, 15×15, and 20×20), each with their own color palette (9 tasks total; Figure D.3). Tasks `SCMu3am` and `SCMtcbq` were generated analogously, using one logical operator combination each (6 tasks total; 3 per SCM).

D.1.2 Program Synthesis

Tasks A single task was sampled from each of four SCMs: `SCMm5ob` (counting), `SCMev5t` (counting), `SCMfwpq` (extension), and `SCMz750` (extension). Example prompts are provided in Figures D.6 and D.7. Prompts were sampled with varying numbers of in-context demonstrations (4, 6, 8) to assess impacts on program correctness.

D.1.3 Causal Discovery with Logical Reasoning

Tasks For each of three tasks sampled from `SCMtcbq` (where logical operators composed *xor* after *and*), five prompts were sampled per level of the PCH. Tasks featured scaling problem complexity with respect to output array size (10×10, 15×15, and 20×20).

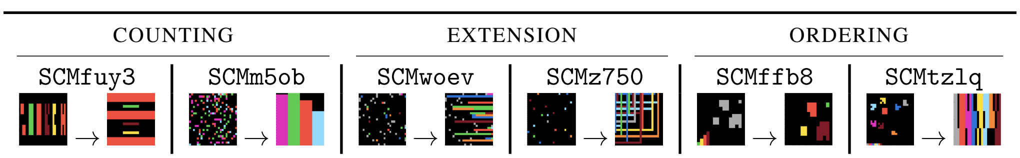

Figure D.2: Counting, extension, and ordering test cases for the counterfactual reasoning experiment.

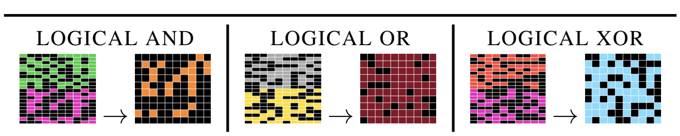

Figure D.3: Samples from CausalARC task `SCMdky5`, where causal functions were *and*, *or*, and *xor*.

Prompt

```
You must solve the following puzzle by discovering the deterministic rule that
maps inputs to outputs. You will then be asked to predict the output for a
counterfactual example. Both the inputs and outputs are 2D grids of colored
pixels. We provide example input-output pairs as demonstration. Grids are
provided as Python arrays. You must output only a single Python array, and do
not explain your reasoning.
Example input-output arrays:
[[0, 4, 4, 0, 4, 0, 0, 4, 0, 4], [4, 4, 0, 0, 4, 0, 4, 4, 0, 4], [0, 4, 4, 0, 0,
4, 4, 4, 0, 4], [4, 4, 4, 4, 0, 0, 4, 0, 4, 0], [4, 0, 0, 0, 0, 0, 0, 4, 4, 0],
[4, 4, 4, 0, 0, 4, 0, 0, 0, 4], [4, 4, 4, 0, 4, 4, 0, 4, 4, 4], [0, 0, 4, 0, 0,
4, 4, 4, 0, 4], [4, 4, 4, 0, 4, 0, 4, 4, 4, 0], [0, 0, 4, 4, 0, 4, 0, 0, 4, 4],
[0, 0, 5, 0, 5, 0, 5, 0, 0, 5], [0, 5, 5, 5, 5, 5, 0, 5, 0, 5], [5, 5, 0, 0, 0,
5, 5, 0, 0, 5], [0, 0, 0, 0, 5, 5, 0, 0, 0, 0], [0, 0, 0, 5, 5, 0, 0, 5, 5, 0],
[5, 0, 0, 5, 5, 5, 0, 0, 5, 5], [5, 0, 0, 0, 5, 5, 0, 0, 5, 0], [0, 5, 5, 5, 5,
0, 5, 0, 5, 0], [5, 0, 0, 0, 5, 0, 5, 5, 5, 0], [5, 0, 0, 5, 0, 5, 5, 0, 5, 0]]
-> [[0, 0, 2, 0, 2, 0, 0, 0, 0, 2], [0, 2, 0, 0, 2, 0, 0, 2, 0, 2], [0, 2, 0, 0,
0, 2, 2, 0, 0, 2], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 2, 2,
0], [2, 0, 0, 0, 0, 2, 0, 0, 0, 2], [2, 0, 0, 0, 2, 2, 0, 0, 2, 0], [0, 0, 2, 0,
0, 0, 2, 0, 0, 0], [2, 0, 0, 0, 2, 0, 2, 2, 2, 0], [0, 0, 0, 2, 0, 2, 0, 0, 2,
0]]
Example input-output arrays:
[[4, 0, 0, 4, 0, 0, 0, 4, 4, 4], [0, 4, 0, 4, 0, 0, 4, 0, 0, 0], [0, 4, 4, 0, 4,
0, 4, 4, 0, 0], [0, 4, 0, 0, 4, 0, 0, 0, 0, 0], [0, 0, 4, 0, 0, 0, 0, 0, 0, 4],
[4, 0, 0, 0, 4, 4, 0, 0, 0, 0], [0, 0, 0, 0, 4, 4, 4, 0, 4, 4], [4, 0, 0, 0, 4,
0, 4, 0, 0, 4], [4, 4, 4, 0, 4, 4, 0, 4, 0, 0], [4, 0, 0, 4, 0, 4, 0, 0, 0, 0],
[5, 5, 5, 0, 5, 0, 5, 0, 5, 5], [0, 5, 5, 0, 5, 5, 0, 5, 0, 5], [5, 0, 5, 5, 5,
0, 0, 0, 0, 0], [5, 0, 0, 5, 5, 0, 5, 0, 5, 5], [5, 5, 5, 5, 5, 5, 0, 5, 5, 5],
[5, 5, 0, 0, 5, 0, 0, 0, 5, 5], [0, 0, 5, 5, 0, 5, 5, 5, 5, 5], [5, 0, 0, 0, 0,
5, 0, 0, 5, 5], [0, 5, 5, 5, 5, 0, 5, 5, 0, 0], [0, 0, 0, 0, 0, 0, 5, 5, 0, 5]]
-> [[2, 0, 0, 0, 0, 0, 0, 0, 2, 2], [0, 2, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 2, 0,
2, 0, 0, 0, 0, 0], [0, 0, 0, 0, 2, 0, 0, 0, 0, 0], [0, 0, 2, 0, 0, 0, 0, 0, 0,
2], [2, 0, 0, 0, 2, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 2, 2, 0, 2, 2], [2, 0, 0, 0,
0, 0, 0, 0, 0, 2], [0, 2, 2, 0, 2, 0, 0, 2, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0,
0]]
Example input-output arrays:
[[0, 0, 0, 4, 0, 0, 0, 0, 0, 4], [0, 4, 0, 0, 4, 0, 0, 0, 4, 0], [4, 4, 0, 4, 0,
4, 4, 4, 4, 0], [4, 0, 0, 0, 4, 4, 0, 4, 0, 4], [4, 0, 0, 0, 0, 0, 4, 0, 0, 4],
[0, 4, 4, 4, 0, 0, 0, 0, 4, 0], [0, 4, 4, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0,
0, 4, 0, 4, 4], [0, 4, 4, 0, 4, 0, 4, 4, 0, 4], [4, 4, 0, 0, 4, 0, 4, 4, 4, 4],
[5, 0, 0, 5, 5, 0, 0, 0, 0, 5], [0, 0, 5, 0, 5, 0, 0, 0, 0, 0], [5, 0, 0, 0, 0,
5, 0, 5, 0, 0], [0, 0, 0, 5, 5, 5, 0, 5, 0, 5], [0, 0, 0, 0, 0, 5, 5, 0, 0, 5],
[5, 5, 5, 0, 0, 5, 5, 0, 0, 5], [0, 5, 5, 5, 0, 0, 5, 5, 0, 5], [0, 5, 5, 0, 0,
0, 0, 5, 5, 5], [5, 5, 0, 5, 5, 5, 0, 5, 0, 0], [0, 5, 5, 0, 5, 5, 0, 5, 5, 0]]
-> [[0, 0, 0, 2, 0, 0, 0, 0, 0, 2], [0, 0, 0, 0, 2, 0, 0, 0, 0, 0], [2, 0, 0, 0,
0, 2, 0, 2, 0, 0], [0, 0, 0, 0, 2, 2, 0, 2, 0, 2], [0, 0, 0, 0, 0, 0, 2, 0, 0,
2], [0, 2, 2, 0, 0, 0, 0, 0, 0, 0], [0, 2, 2, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0,
0, 0, 0, 0, 2, 2], [0, 2, 0, 0, 2, 0, 0, 2, 0, 0], [0, 2, 0, 0, 2, 0, 0, 2, 2,
0]]
Counterfactual: Now imagine that we intervened on the previous input by changing
some colors.
[[0, 0, 0, 8, 0, 0, 0, 0, 0, 8], [0, 8, 0, 0, 8, 0, 0, 0, 8, 0], [8, 8, 0, 8, 0,
8, 8, 8, 8, 0], [8, 0, 0, 0, 8, 8, 0, 8, 0, 8], [8, 0, 0, 0, 0, 0, 8, 0, 0, 8],
[0, 8, 8, 8, 0, 0, 0, 0, 8, 0], [0, 8, 8, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0,
0, 8, 0, 8, 8], [0, 8, 8, 0, 8, 0, 8, 8, 0, 8], [8, 8, 0, 0, 8, 0, 8, 8, 8, 8],
[5, 0, 0, 5, 5, 0, 0, 0, 0, 5], [0, 0, 5, 0, 5, 0, 0, 0, 0, 0], [5, 0, 0, 0, 0,
5, 0, 5, 0, 0], [0, 0, 0, 5, 5, 5, 0, 5, 0, 5], [0, 0, 0, 0, 0, 5, 5, 0, 0, 5],
[5, 5, 5, 0, 0, 5, 5, 0, 0, 5], [0, 5, 5, 5, 0, 0, 5, 5, 0, 5], [0, 5, 5, 0, 0,
0, 0, 5, 5, 5], [5, 5, 0, 5, 5, 5, 0, 5, 0, 0], [0, 5, 5, 0, 5, 5, 0, 5, 5, 0]]
->
```

Figure D.4: L1 prompt for counterfactual reasoning.

Prompt

```
You must solve the following puzzle by discovering the deterministic rule
that maps inputs to outputs.  You will then be asked to predict the output
for a counterfactual example.  Both the inputs and outputs are 2D grids of
colored pixels.  We provide example input-output pairs along with counterfactual
examples, which represent interventions on the original examples.  Grids are
provided as Python arrays.  You must output only a single Python array, and do
not explain your reasoning.
Example input-output arrays:
[[4, 0, 0, 4, 0, 0, 0, 4, 4, 4], [0, 4, 0, 4, 0, 0, 4, 0, 0, 0], [0, 4, 4, 0, 4,
0, 4, 4, 0, 0], [0, 4, 0, 0, 4, 0, 0, 0, 0, 0], [0, 0, 4, 0, 0, 0, 0, 0, 0, 4],
[4, 0, 0, 0, 4, 4, 0, 0, 0, 0], [0, 0, 0, 0, 4, 4, 4, 0, 4, 4], [4, 0, 0, 0, 4,
0, 4, 0, 0, 4], [4, 4, 4, 0, 4, 4, 0, 4, 0, 0], [4, 0, 0, 4, 0, 4, 0, 0, 0, 0],
[5, 5, 5, 0, 5, 0, 5, 0, 5, 5], [0, 5, 5, 0, 5, 5, 0, 5, 0, 5], [5, 0, 5, 5, 5,
0, 0, 0, 0, 0], [5, 0, 0, 5, 5, 0, 5, 0, 5, 5], [5, 5, 5, 5, 5, 5, 0, 5, 5, 5],
[5, 5, 0, 0, 5, 0, 0, 0, 5, 5], [0, 0, 5, 5, 0, 5, 5, 5, 5, 5], [5, 0, 0, 0, 0,
5, 0, 0, 5, 5], [0, 5, 5, 5, 5, 0, 5, 5, 0, 0], [0, 0, 0, 0, 0, 0, 5, 5, 0, 5]]
-> [[2, 0, 0, 0, 0, 0, 0, 0, 2, 2], [0, 2, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 2, 0,
2, 0, 0, 0, 0, 0], [0, 0, 0, 0, 2, 0, 0, 0, 0, 0], [0, 0, 2, 0, 0, 0, 0, 0, 0,
2], [2, 0, 0, 0, 2, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 2, 2, 0, 2, 2], [2, 0, 0, 0,
0, 0, 0, 0, 0, 2], [0, 2, 2, 0, 2, 0, 0, 2, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0,
0]]
Counterfactual:  Now imagine that we intervened on the previous input by fixing
some values.
[[4, 0, 0, 4, 0, 0, 0, 4, 4, 4], [0, 4, 0, 4, 0, 0, 4, 0, 0, 0], [0, 4, 4, 0, 4,
0, 4, 4, 0, 0], [0, 4, 0, 0, 4, 0, 0, 0, 0, 0], [0, 0, 4, 0, 0, 0, 0, 0, 0, 4],
[4, 0, 0, 0, 4, 4, 0, 0, 0, 0], [0, 0, 0, 0, 4, 4, 4, 0, 4, 4], [4, 0, 0, 0, 4,
0, 4, 0, 0, 4], [4, 4, 4, 0, 4, 4, 0, 4, 0, 0], [4, 0, 0, 4, 0, 4, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0,
0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0,
0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
-> [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0,
0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0,
0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0,
0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0,
0]]
Example input-output arrays:
[[0, 0, 0, 4, 0, 0, 0, 0, 0, 4], [0, 4, 0, 0, 4, 0, 0, 0, 4, 0], [4, 4, 0, 4, 0,
4, 4, 4, 4, 0], [4, 0, 0, 0, 4, 4, 0, 4, 0, 4], [4, 0, 0, 0, 0, 0, 4, 0, 0, 4],
[0, 4, 4, 4, 0, 0, 0, 0, 4, 0], [0, 4, 4, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0,
0, 4, 0, 4, 4], [0, 4, 4, 0, 4, 0, 4, 4, 0, 4], [4, 4, 0, 0, 4, 0, 4, 4, 4, 4],
[5, 0, 0, 5, 5, 0, 0, 0, 0, 5], [0, 0, 5, 0, 5, 0, 0, 0, 0, 0], [5, 0, 0, 0, 0,
5, 0, 5, 0, 0], [0, 0, 0, 5, 5, 5, 0, 5, 0, 5], [0, 0, 0, 0, 0, 5, 5, 0, 0, 5],
[5, 5, 5, 0, 0, 5, 5, 0, 0, 5], [0, 5, 5, 5, 0, 0, 5, 5, 0, 5], [0, 5, 5, 0, 0,
0, 0, 5, 5, 5], [5, 5, 0, 5, 5, 5, 0, 5, 0, 0], [0, 5, 5, 0, 5, 5, 0, 5, 5, 0]]
-> [[0, 0, 0, 2, 0, 0, 0, 0, 0, 2], [0, 0, 0, 0, 2, 0, 0, 0, 0, 0], [2, 0, 0, 0,
0, 2, 0, 2, 0, 0], [0, 0, 0, 0, 2, 2, 0, 2, 0, 2], [0, 0, 0, 0, 0, 0, 2, 0, 0,
2], [0, 2, 2, 0, 0, 0, 0, 0, 0, 0], [0, 2, 2, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0,
0, 0, 0, 0, 2, 2], [0, 2, 0, 0, 2, 0, 0, 2, 0, 0], [0, 2, 0, 0, 2, 0, 0, 2, 2,
0]]
Counterfactual:  Now imagine that we intervened on the previous input by changing
some colors.
[[0, 0, 0, 8, 0, 0, 0, 0, 0, 8], [0, 8, 0, 0, 8, 0, 0, 0, 8, 0], [8, 8, 0, 8, 0,
8, 8, 8, 8, 0], [8, 0, 0, 0, 8, 8, 0, 8, 0, 8], [8, 0, 0, 0, 0, 0, 8, 0, 0, 8],
[0, 8, 8, 8, 0, 0, 0, 0, 8, 0], [0, 8, 8, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0,
0, 8, 0, 8, 8], [0, 8, 8, 0, 8, 0, 8, 8, 0, 8], [8, 8, 0, 0, 8, 0, 8, 8, 8, 8],
[5, 0, 0, 5, 5, 0, 0, 0, 0, 5], [0, 0, 5, 0, 5, 0, 0, 0, 0, 0], [5, 0, 0, 0, 0,
5, 0, 5, 0, 0], [0, 0, 0, 5, 5, 5, 0, 5, 0, 5], [0, 0, 0, 0, 0, 5, 5, 0, 0, 5],
[5, 5, 5, 0, 0, 5, 5, 0, 0, 5], [0, 5, 5, 5, 0, 0, 5, 5, 0, 5], [0, 5, 5, 0, 0,
0, 0, 5, 5, 5], [5, 5, 0, 5, 5, 5, 0, 5, 0, 0], [0, 5, 5, 0, 5, 5, 0, 5, 5, 0]]
->
```

Figure D.5: L3 prompt for counterfactual reasoning.

Prompt

```
You must solve the following puzzle by discovering the deterministic rule that
maps inputs to outputs.  Both the inputs and outputs are 2D Python arrays of
colored pixels.  We provide example input-output pairs as demonstration.  To
solve the problem, express the deterministic rule as a Python program.  Do not
explain your reasoning, and only output a single Python program.
Example input-output arrays:
[[0, 0, 0, 0, 0, 9, 0, 4, 0, 3], [0, 4, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 4,
0, 2, 0, 0, 0], [0, 0, 0, 0, 0, 9, 0, 0, 2, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 3],
[0, 9, 0, 0, 0, 4, 0, 0, 0, 2], [0, 0, 0, 0, 0, 9, 0, 2, 4, 0], [3, 0, 3, 0, 3,
0, 0, 0, 3, 0], [0, 0, 4, 3, 0, 0, 3, 0, 4, 0], [9, 3, 0, 0, 0, 0, 2, 0, 0, 0]]
-> [[0, 0, 0, 3], [0, 0, 0, 3], [4, 0, 0, 3], [4, 0, 0, 3], [4, 2, 9, 3], [4, 2,
9, 3], [4, 2, 9, 3], [4, 2, 9, 3], [4, 2, 9, 3]]
Example input-output arrays:
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [9, 0, 0, 2, 0, 0, 9, 0, 0, 0], [0, 0, 0, 0, 0,
0, 2, 0, 9, 0], [9, 4, 0, 0, 9, 3, 0, 0, 0, 0], [0, 0, 0, 2, 0, 0, 2, 3, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 9, 4, 0, 9, 0, 0, 0, 2, 9], [0, 4, 0, 0, 0,
2, 0, 0, 0, 0], [0, 0, 3, 0, 9, 0, 9, 0, 9, 4], [0, 0, 9, 0, 4, 0, 0, 0, 3, 0]]
-> [[0, 0, 9, 0], [0, 0, 9, 0], [0, 0, 9, 0], [0, 0, 9, 0], [0, 0, 9, 0], [0, 0,
9, 0], [0, 2, 9, 0], [4, 2, 9, 0], [4, 2, 9, 3], [4, 2, 9, 3], [4, 2, 9, 3], [4,
2, 9, 3]]
Example input-output arrays:
[[0, 0, 0, 0, 0, 3, 0, 0, 0, 3], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 9, 0, 0, 3,
0, 0, 0, 0, 0], [0, 9, 3, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 4, 0, 0],
[3, 0, 0, 0, 0, 3, 0, 9, 0, 0], [0, 0, 4, 0, 0, 9, 0, 0, 9, 4], [0, 3, 0, 0, 0,
0, 0, 0, 0, 0], [9, 0, 4, 3, 2, 9, 0, 0, 0, 0], [4, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
-> [[0, 0, 0, 3], [0, 0, 9, 3], [0, 0, 9, 3], [4, 0, 9, 3], [4, 0, 9, 3], [4, 0,
9, 3], [4, 0, 9, 3], [4, 2, 9, 3]]
Example input-output arrays:
[[0, 0, 2, 0, 0, 0, 0, 0, 0, 0], [0, 0, 9, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 2,
0, 0, 0, 0, 0], [0, 9, 2, 2, 2, 0, 0, 0, 0, 9], [0, 0, 2, 0, 0, 0, 0, 0, 0, 0],
[0, 4, 3, 0, 0, 0, 0, 0, 3, 0], [0, 0, 0, 2, 3, 0, 0, 0, 0, 0], [0, 0, 4, 9, 2,
0, 0, 0, 4, 4], [0, 0, 0, 0, 4, 0, 0, 0, 0, 0], [0, 0, 4, 0, 0, 0, 0, 9, 3, 9]]
-> [[0, 2, 0, 0], [0, 2, 0, 0], [4, 2, 9, 0], [4, 2, 9, 0], [4, 2, 9, 3], [4, 2,
9, 3], [4, 2, 9, 3], [4, 2, 9, 3]]
Example input-output arrays:
[[0, 9, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 3, 0, 9, 2, 0], [0, 0, 0, 0, 0,
2, 0, 0, 0, 0], [0, 4, 2, 0, 0, 0, 0, 9, 0, 0], [0, 0, 0, 4, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 2, 0, 0, 0], [0, 0, 0, 2, 4, 0, 0, 3, 0, 0], [3, 2, 0, 4, 0,
3, 0, 2, 0, 0], [0, 3, 0, 0, 4, 0, 9, 9, 4, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
-> [[0, 2, 0, 0], [4, 2, 0, 0], [4, 2, 9, 3], [4, 2, 9, 3], [4, 2, 9, 3], [4, 2,
9, 3], [4, 2, 9, 3]]
Example input-output arrays:
[[0, 0, 0, 0, 0, 3, 0, 0, 0, 3], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 9, 0, 0, 3,
0, 0, 0, 0, 0], [0, 9, 3, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 4, 0, 0],
[3, 0, 0, 0, 0, 3, 0, 9, 0, 0], [0, 0, 4, 0, 0, 9, 0, 0, 9, 4], [0, 3, 0, 0, 0,
0, 0, 0, 0, 0], [9, 0, 4, 3, 2, 9, 0, 0, 0, 0], [4, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
-> [[0, 0, 0, 3], [0, 0, 9, 3], [0, 0, 9, 3], [4, 0, 9, 3], [4, 0, 9, 3], [4, 0,
9, 3], [4, 0, 9, 3], [4, 2, 9, 3]]
```

Figure D.6: L1 prompt for program synthesis with six in-context examples.

Prompt

```
You must solve the following puzzle by discovering the deterministic rule that
maps inputs to outputs.  Both the inputs and outputs are 2D Python arrays of
colored pixels.  We provide example input-output pairs along with counterfactual
examples, which represent interventions on the original examples.  To solve the
problem, express the deterministic rule as a Python program.  Do not explain your
reasoning, and only output a single Python program.
Example input-output arrays:
[[4, 3, 0, 0, 0, 9, 9, 9, 0, 0], [0, 0, 0, 3, 3, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0,
0, 2, 0, 9, 0], [0, 3, 0, 0, 0, 0, 0, 0, 2, 0], [0, 0, 0, 2, 0, 0, 0, 0, 9, 0],
[0, 0, 0, 0, 0, 0, 9, 2, 0, 0], [0, 2, 0, 0, 0, 0, 0, 3, 2, 4], [4, 0, 0, 0, 0,
0, 0, 9, 0, 0], [0, 0, 3, 0, 0, 0, 0, 2, 0, 0], [0, 0, 0, 9, 0, 0, 3, 0, 0, 2]]
-> [[0, 2, 9, 0], [0, 2, 9, 3], [0, 2, 9, 3], [0, 2, 9, 3], [0, 2, 9, 3], [4, 2,
9, 3], [4, 2, 9, 3], [4, 2, 9, 3]]
Counterfactual:  Now imagine that we intervened on the previous input by rotating
or flipping it.
[[0, 0, 9, 0, 4, 0, 0, 0, 3, 0], [0, 0, 3, 0, 9, 0, 9, 0, 9, 4], [0, 4, 0, 0, 0,
2, 0, 0, 0, 0], [0, 9, 4, 0, 9, 0, 0, 0, 2, 9], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 2, 0, 0, 2, 3, 0, 0], [9, 4, 0, 0, 9, 3, 0, 0, 0, 0], [0, 0, 0, 0, 0,
0, 2, 0, 9, 0], [9, 0, 0, 2, 0, 0, 9, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
-> [[0, 0, 9, 0], [0, 0, 9, 0], [0, 0, 9, 0], [0, 0, 9, 0], [0, 0, 9, 0], [0, 0,
9, 0], [0, 2, 9, 0], [4, 2, 9, 0], [4, 2, 9, 3], [4, 2, 9, 3], [4, 2, 9, 3], [4,
2, 9, 3]]
Example input-output arrays:
[[0, 0, 0, 0, 0, 0, 4, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 4, 0], [0, 0, 0, 2, 0,
0, 0, 0, 0, 0], [0, 4, 0, 0, 0, 9, 0, 0, 4, 0], [0, 0, 0, 0, 0, 3, 0, 0, 3, 0],
[0, 0, 0, 0, 0, 0, 4, 2, 0, 0], [0, 2, 0, 0, 0, 0, 3, 0, 0, 0], [0, 0, 2, 0, 0,
0, 0, 0, 0, 0], [0, 3, 0, 0, 0, 0, 0, 4, 0, 0], [0, 0, 0, 0, 0, 9, 4, 0, 0, 0]]
-> [[4, 0, 0, 0], [4, 0, 0, 0], [4, 0, 0, 0], [4, 2, 0, 3], [4, 2, 0, 3], [4, 2,
9, 3], [4, 2, 9, 3]]
Counterfactual:  Now imagine that we intervened on the previous input by changing
some colors.
[[7, 7, 7, 7, 7, 7, 4, 7, 7, 7], [7, 7, 7, 7, 7, 7, 7, 7, 4, 7], [7, 7, 7, 2, 7,
7, 7, 7, 7, 7], [7, 4, 7, 7, 7, 9, 7, 7, 4, 7], [7, 7, 7, 7, 7, 3, 7, 7, 3, 7],
[7, 7, 7, 7, 7, 7, 4, 2, 7, 7], [7, 2, 7, 7, 7, 7, 3, 7, 7, 7], [7, 7, 2, 7, 7,
7, 7, 7, 7, 7], [7, 3, 7, 7, 7, 7, 7, 4, 7, 7], [7, 7, 7, 7, 7, 9, 4, 7, 7, 7]]
-> [[4, 7, 7, 7], [4, 7, 7, 7], [4, 7, 7, 7], [4, 2, 7, 3], [4, 2, 7, 3], [4, 2,
9, 3], [4, 2, 9, 3]]
Example input-output arrays:
[[0, 0, 0, 0, 0, 0, 0, 3, 0, 0], [4, 0, 0, 0, 0, 9, 3, 0, 0, 0], [0, 0, 0, 0, 0,
0, 0, 4, 0, 0], [0, 0, 0, 0, 2, 0, 4, 0, 0, 0], [0, 3, 0, 0, 4, 0, 0, 0, 2, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 2, 4, 2, 0, 0, 0], [0, 2, 0, 9, 3,
0, 9, 0, 3, 0], [0, 0, 0, 0, 0, 0, 0, 4, 0, 0], [0, 3, 4, 0, 4, 9, 4, 0, 0, 0]]
-> [[4, 0, 0, 0], [4, 0, 0, 0], [4, 0, 0, 0], [4, 0, 0, 3], [4, 2, 0, 3], [4, 2,
9, 3], [4, 2, 9, 3], [4, 2, 9, 3], [4, 2, 9, 3]]
Counterfactual:  Now imagine that we intervened on the previous input by rotating
or flipping it.
[[0, 4, 9, 0, 9, 0, 0, 0, 2, 0], [0, 0, 0, 0, 4, 0, 2, 0, 0, 0], [0, 0, 0, 2, 0,
0, 0, 0, 0, 0], [0, 0, 0, 4, 0, 0, 0, 0, 0, 0], [0, 0, 9, 0, 9, 0, 0, 0, 0, 4],
[9, 0, 0, 0, 2, 0, 0, 4, 9, 4], [0, 0, 0, 0, 0, 0, 0, 2, 0, 0], [0, 0, 9, 0, 0,
3, 0, 3, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 2, 0, 0, 0, 0, 3, 4, 0, 0]]
-> [[4, 0, 9, 0], [4, 2, 9, 0], [4, 2, 9, 0], [4, 2, 9, 0], [4, 2, 9, 3], [4, 2,
9, 3], [4, 2, 9, 3]]
```

Figure D.7: L3 prompt for program synthesis with six in-context examples.

Prompt

```
You must solve the following causal discovery problem, where the cells in an
input array are causal parents of cells in an output array.  Both the inputs and
outputs are 2D Python arrays of colored pixels.  We provide example input-output
pairs as demonstration.  You must predict the causal function(s) that relate
parent cells in the input to their children in the output.  Be concise:  do not
explain your reasoning, and start your answer with ‘The logical operators are’.
Example input-output arrays:
[[4, 0, 4, 0, 0, 4, 4, 4, 4, 0], [4, 4, 0, 4, 4, 4, 0, 4, 0, 0], [0, 0, 4, 0, 0,
0, 4, 4, 4, 0], [4, 4, 4, 0, 4, 4, 0, 0, 0, 0], [4, 4, 0, 0, 4, 0, 0, 4, 4, 0],
[0, 0, 4, 4, 4, 0, 0, 4, 4, 4], [0, 4, 4, 0, 4, 4, 4, 4, 4, 0], [4, 0, 4, 4, 0,
0, 0, 4, 0, 0], [4, 4, 4, 4, 0, 4, 0, 4, 0, 4], [4, 4, 0, 4, 4, 0, 0, 0, 0, 4],
[0, 5, 0, 5, 0, 0, 5, 5, 5, 0], [0, 0, 5, 0, 5, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0,
5, 5, 5, 5, 5], [5, 0, 0, 5, 5, 5, 5, 0, 5, 5], [5, 0, 0, 5, 0, 5, 0, 0, 0, 0],
[0, 5, 0, 5, 5, 0, 5, 0, 5, 5], [0, 0, 5, 0, 0, 5, 0, 5, 5, 5], [5, 0, 5, 0, 0,
0, 5, 5, 0, 0], [0, 0, 5, 5, 0, 0, 0, 0, 0, 0], [0, 5, 5, 5, 0, 5, 0, 0, 0, 5],
[2, 0, 0, 0, 2, 0, 2, 0, 0, 2], [2, 0, 2, 2, 2, 2, 0, 0, 0, 0], [2, 0, 2, 2, 0,
2, 2, 0, 2, 2], [2, 0, 0, 2, 2, 0, 0, 2, 2, 0], [0, 0, 2, 2, 2, 2, 2, 2, 0, 0],
[0, 2, 2, 2, 2, 0, 2, 0, 0, 0], [0, 0, 2, 0, 2, 0, 0, 2, 2, 0], [0, 0, 0, 2, 2,
0, 0, 2, 2, 2], [2, 2, 0, 0, 2, 2, 0, 0, 2, 2], [0, 2, 0, 2, 0, 0, 0, 2, 2, 2]]
-> [[1, 0, 0, 0, 1, 0, 0, 1, 1, 1], [1, 0, 1, 1, 0, 1, 0, 0, 0, 0], [1, 0, 1, 1,
0, 1, 0, 1, 0, 1], [0, 0, 0, 1, 0, 1, 0, 1, 1, 0], [1, 0, 1, 1, 1, 1, 1, 1, 0,
0], [0, 1, 1, 0, 0, 0, 1, 0, 1, 1], [0, 0, 0, 0, 1, 1, 0, 0, 0, 0], [1, 0, 1, 1,
1, 0, 0, 0, 1, 1], [1, 1, 1, 1, 1, 1, 0, 0, 1, 1], [0, 0, 0, 0, 0, 0, 0, 1, 1,
0]]
Example input-output arrays:
[[0, 4, 4, 4, 0, 4, 4, 0, 4, 4], [0, 4, 4, 4, 0, 0, 4, 4, 4, 4], [0, 4, 0, 4, 4,
0, 0, 4, 4, 0], [0, 0, 0, 0, 0, 4, 0, 0, 0, 4], [0, 4, 4, 0, 0, 4, 0, 4, 4, 4],
[0, 4, 0, 4, 4, 4, 4, 0, 0, 0], [0, 0, 0, 0, 4, 4, 4, 0, 0, 0], [0, 0, 0, 4, 0,
0, 0, 4, 0, 0], [0, 4, 0, 0, 4, 0, 4, 4, 0, 0], [4, 0, 4, 4, 4, 0, 4, 4, 0, 4],
[0, 0, 0, 5, 5, 5, 0, 0, 5, 5], [0, 5, 0, 5, 0, 5, 0, 5, 0, 5], [0, 5, 5, 5, 5,
5, 0, 5, 0, 0], [0, 5, 5, 0, 0, 0, 5, 0, 5, 5], [0, 0, 5, 0, 5, 5, 5, 5, 0, 0],
[0, 0, 5, 0, 5, 0, 5, 5, 5, 5], [5, 5, 5, 0, 0, 0, 0, 0, 5, 5], [5, 5, 0, 0, 5,
0, 5, 0, 5, 5], [0, 0, 5, 0, 0, 0, 0, 5, 5, 0], [5, 5, 5, 0, 5, 0, 0, 5, 5, 5],
[0, 0, 2, 0, 0, 2, 0, 0, 0, 0], [2, 2, 2, 2, 0, 2, 2, 0, 0, 2], [0, 0, 2, 0, 0,
2, 0, 2, 2, 2], [2, 0, 0, 2, 2, 0, 2, 0, 2, 2], [0, 2, 2, 2, 0, 2, 0, 2, 2, 2],
[0, 2, 0, 0, 2, 2, 2, 2, 0, 0], [2, 0, 0, 0, 0, 0, 0, 2, 0, 2], [2, 2, 0, 2, 0,
2, 2, 2, 0, 2], [2, 0, 0, 2, 0, 0, 0, 2, 2, 2], [2, 0, 0, 2, 2, 0, 0, 2, 2, 0]]
-> [[0, 0, 1, 1, 0, 0, 0, 0, 1, 1], [1, 0, 1, 0, 0, 1, 1, 1, 0, 0], [0, 1, 1, 1,
1, 1, 0, 0, 1, 1], [1, 0, 0, 1, 1, 0, 1, 0, 1, 0], [0, 1, 0, 1, 0, 0, 0, 0, 1,
1], [0, 1, 0, 0, 0, 1, 0, 1, 0, 0], [1, 0, 0, 0, 0, 0, 0, 1, 0, 1], [1, 1, 0, 1,
0, 1, 1, 1, 0, 1], [1, 0, 0, 1, 0, 0, 0, 0, 1, 1], [0, 0, 1, 1, 0, 0, 0, 0, 1,
1]]
...
```

Figure D.8: Excerpt of L1 prompt for logical reasoning. Four in-context examples were provided.

Prompt

```
You must solve the following causal discovery problem, where the cells in an
input array are causal parents of cells in an output array.  Both the inputs and
outputs are 2D Python arrays of colored pixels.  We provide example input-output
pairs along with counterfactual examples, which represent interventions on the
original examples.  You must predict the causal function(s) that relate parent
cells in the input to their children in the output.  Be concise:  do not explain
your reasoning, and start your answer with ‘The logical operators are’.
Example input-output arrays:
[[0, 4, 0, 0, 0, 4, 4, 0, 0, 0], [0, 4, 4, 0, 4, 0, 0, 4, 4, 4], [0, 0, 0, 0, 0,
4, 4, 4, 4, 4], [4, 4, 0, 4, 0, 0, 0, 4, 0, 0], [4, 0, 0, 4, 0, 0, 0, 4, 0, 0],
[0, 4, 0, 4, 0, 4, 0, 0, 4, 0], [0, 4, 0, 0, 0, 4, 4, 0, 4, 4], [4, 4, 4, 0, 4,
4, 4, 0, 0, 4], [0, 4, 4, 0, 0, 4, 0, 4, 4, 4], [4, 4, 4, 0, 0, 0, 0, 4, 0, 4],
[5, 0, 5, 5, 5, 5, 5, 0, 5, 0], [5, 5, 5, 5, 0, 5, 0, 0, 0, 5], [0, 0, 0, 0, 0,
0, 0, 5, 5, 5], [0, 0, 0, 0, 5, 5, 0, 0, 5, 0], [0, 0, 0, 0, 0, 5, 5, 5, 5, 5],
[5, 5, 5, 0, 0, 0, 5, 0, 0, 5], [0, 0, 0, 5, 0, 5, 0, 0, 0, 5], [5, 5, 0, 0, 0,
0, 0, 5, 5, 0], [5, 5, 5, 0, 5, 0, 5, 0, 0, 5], [0, 5, 5, 0, 5, 0, 5, 0, 5, 5],
[2, 0, 0, 2, 0, 0, 0, 2, 0, 2], [2, 2, 0, 2, 2, 2, 2, 2, 2, 2], [2, 0, 0, 0, 0,
2, 0, 0, 2, 0], [2, 2, 2, 0, 2, 0, 0, 2, 0, 2], [2, 2, 0, 0, 0, 0, 2, 0, 2, 0],
[2, 2, 2, 0, 2, 0, 2, 2, 0, 0], [0, 2, 0, 0, 0, 2, 2, 0, 2, 0], [2, 0, 0, 2, 2,
0, 0, 2, 2, 0], [0, 2, 0, 0, 2, 2, 2, 2, 2, 0], [0, 2, 0, 2, 0, 2, 0, 2, 2, 2]]
-> [[1, 0, 0, 1, 0, 1, 1, 1, 0, 1], [1, 0, 1, 1, 1, 1, 1, 1, 1, 0], [1, 0, 0, 0,
0, 1, 0, 1, 0, 1], [1, 1, 1, 0, 1, 0, 0, 1, 0, 1], [1, 1, 0, 0, 0, 0, 1, 1, 1,
0], [1, 0, 1, 0, 1, 0, 1, 1, 0, 0], [0, 1, 0, 0, 0, 0, 1, 0, 1, 1], [0, 1, 0, 1,
1, 0, 0, 1, 1, 0], [0, 0, 1, 0, 1, 1, 1, 1, 1, 1], [0, 0, 1, 1, 0, 1, 0, 1, 1,
0]]
Counterfactual:  Now imagine that we intervened on the original input by changing
some colors.
[[4, 4, 4, 4, 0, 0, 0, 4, 4, 4], [0, 0, 0, 4, 0, 0, 4, 4, 4, 0], [4, 0, 0, 4, 4,
4, 0, 4, 4, 0], [0, 0, 0, 0, 4, 0, 0, 0, 0, 4], [4, 0, 0, 0, 0, 4, 0, 4, 4, 0],
[0, 0, 4, 4, 0, 0, 0, 4, 4, 0], [4, 0, 4, 0, 0, 4, 4, 4, 0, 0], [0, 4, 0, 0, 0,
0, 0, 4, 4, 0], [0, 0, 0, 0, 4, 0, 4, 0, 0, 4], [4, 0, 4, 0, 4, 4, 0, 4, 4, 4],
[0, 0, 6, 0, 6, 6, 6, 6, 6, 6], [6, 0, 0, 0, 0, 6, 6, 6, 0, 0], [6, 0, 6, 6, 0,
0, 6, 0, 0, 6], [6, 0, 0, 6, 6, 0, 0, 6, 0, 6], [6, 0, 0, 6, 6, 0, 6, 0, 6, 0],
[6, 0, 6, 6, 0, 0, 6, 6, 6, 6], [0, 6, 0, 6, 0, 6, 0, 6, 0, 0], [0, 6, 6, 0, 6,
6, 0, 0, 6, 0], [6, 0, 6, 6, 0, 6, 0, 0, 0, 6], [0, 0, 0, 6, 0, 6, 0, 6, 6, 6],
[0, 0, 2, 2, 2, 2, 0, 0, 2, 2], [0, 2, 2, 0, 2, 2, 2, 0, 2, 2], [2, 2, 2, 2, 2,
0, 0, 0, 0, 0], [0, 2, 0, 0, 0, 0, 2, 0, 0, 2], [2, 0, 0, 0, 0, 0, 0, 0, 0, 2],
[0, 2, 0, 0, 0, 0, 2, 2, 2, 2], [2, 0, 2, 0, 2, 0, 2, 0, 2, 0], [0, 0, 2, 2, 0,
2, 0, 0, 0, 0], [0, 2, 2, 0, 0, 0, 2, 0, 2, 0], [0, 0, 2, 2, 2, 2, 0, 2, 0, 0]]
-> [[0, 0, 0, 1, 1, 1, 0, 1, 0, 0], [0, 1, 1, 0, 1, 1, 0, 1, 1, 1], [0, 1, 1, 0,
1, 0, 0, 0, 0, 0], [0, 1, 0, 0, 1, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 1,
1], [0, 1, 1, 1, 0, 0, 1, 0, 0, 1], [1, 0, 1, 0, 1, 1, 1, 1, 1, 0], [0, 1, 1, 1,
0, 1, 0, 0, 1, 0], [0, 1, 1, 0, 0, 0, 1, 0, 1, 1], [0, 0, 1, 1, 1, 0, 0, 0, 1,
1]]
...
```

Figure D.9: Excerpt of L3 prompt for logical reasoning. Four in-context examples were provided.

Response

```
# For ARC-style input/output:
if __name__ == "__main__":
  import sys, json
  data = json.load(sys.stdin)
  inp = data["input"]
  out = solve(inp)
  print(json.dumps(out))
```

Figure D.10: Evidence that o4-mini may have been fine-tuned for ARC-like program synthesis ("`For ARC-style input/output:... out = solve(inp)`"), excerpted from a response to a CausalARC query that did not mention ARC in-context.

Response

```
Replicate 1: The logical operators are: For each output cell at '(row, col)',
let 'A = (input[row][col] != 0)', 'B = (input[row + 10][col] != 0)', and 'C =
(input[row + 20][col] != 0)'. The output cell 'output[row][col]' is '1' if '(A
AND B) XOR C' is true, and '0' otherwise.


Replicate 2: The logical operators are '((x AND y) XOR z)', where 'x', 'y', and
'z' are boolean values indicating if the corresponding input cell (from the first,
second, and third 15x15 input arrays, respectively) is non-zero. The output cell
is '2' if the expression is true, and '0' if false.


Replicate 3: The logical operators are XOR and AND. For each output cell
'O[r][c]', its value is determined by three input cells: 'P1 = I[r][c]', 'P2
= I[r+10][c]', and 'P3 = I[r+20][c]'. The output value is '1' if '(P3 != 0) XOR
((P1 != 0) AND (P2 != 0))' is true, and '0' otherwise.
```

Figure D.11: Three correct replicate responses from Gemini 2.5 Flash for the causal discovery with logical reasoning task. The causal parents and logical operators were correctly identified.

Response

```
Replicate 1: The logical operators are AND and XOR.


Replicate 3: The logical operators are AND between the first and second input
grids, then XOR with the third input grid.


Replicate 4: The logical operators are AND and XOR, applied pointwise as (color4
AND color5) XOR color2.
```

Figure D.12: Three correct replicate responses from Grok 4 for the causal discovery with logical reasoning task. Logical operators were correctly identified and causal parents were explicitly declared in some cases.

D.2 Experimental Results

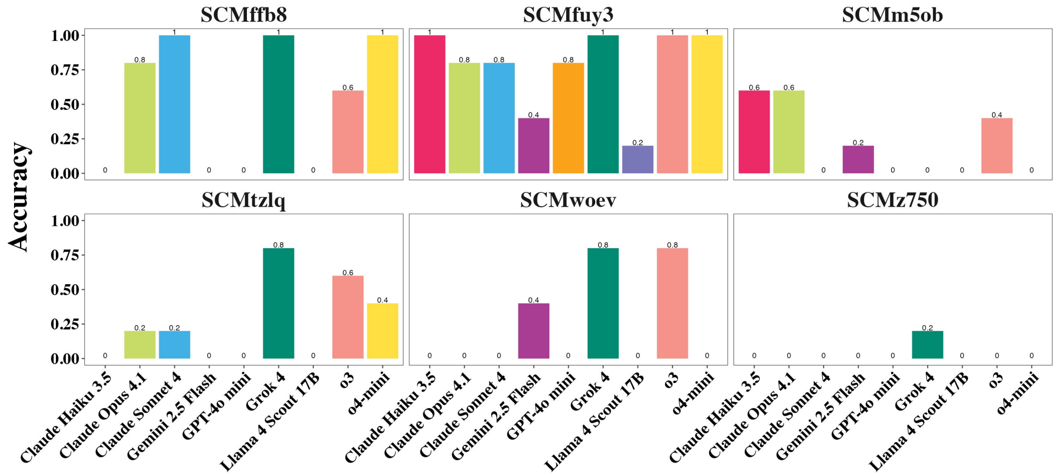

Figure D.13: **Counterfactual reasoning** on counting, extension, and ordering tasks (Figure D.2). Scores are over five random prompt samples.

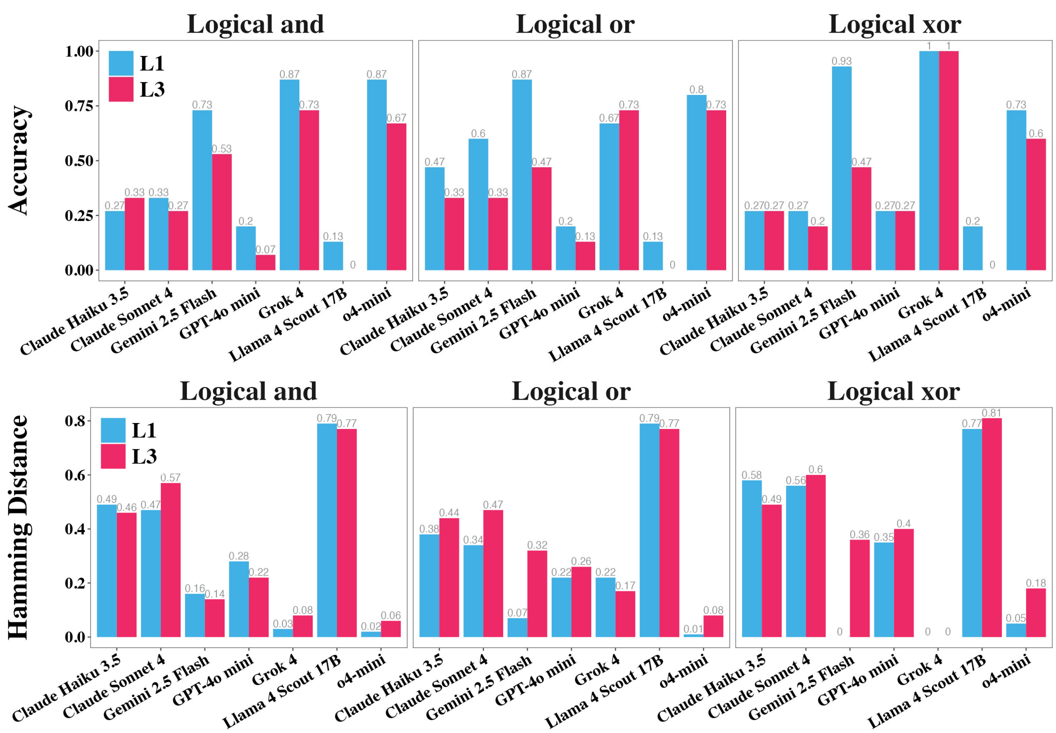

Figure D.14: **Counterfactual reasoning** results on variants of CausalARC task `SCMdky5`. Scores are over five random prompt samples. Raw data are in Table D.2. Metrics are accuracy (fraction tasks answered correctly; values between 0 and 1; higher is better) and Hamming distance (values between 0 and 1; lower is better). Accurate prediction corresponds to a Hamming distance of 0.

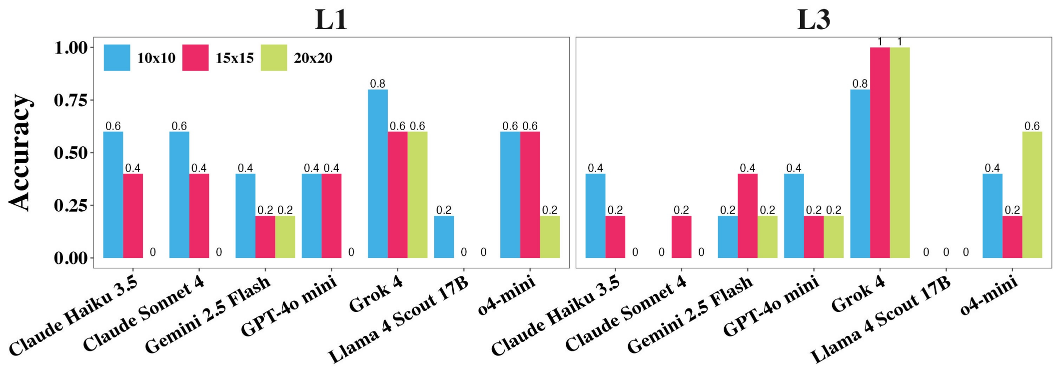

Figure D.15: **Counterfactual reasoning** results on variants of CausalARC task `SCMtcbq`, where logical operators compose (*xor* after *and*). Accuracy was measured with respect to scaling the dimensionality of the output array (10×10, 15×15, 20×20). Scores are over five random prompt samples. Raw data are provided in Table D.3.

	LOGICAL AND				LOGICAL OR				LOGICAL XOR			
	L1		L3		L1		L3		L1		L3	
	✓↑	*HD* ↓	✓↑	*HD* ↓	✓↑	*HD* ↓	✓↑	*HD* ↓	✓↑	*HD* ↓	✓↑	*HD* ↓
GPT-4o mini	0.20	0.28 (0.24)	0.07	0.22 (0.15)	0.20	0.22 (0.16)	0.13	0.26 (0.17)	0.27	0.35 (0.23)	0.27	0.4 (0.26)
o4-mini	**0.87**	**0.02** (0.07)	0.67	**0.06** (0.12)	0.80	**0.01** (0.03)	**0.73**	**0.08** (0.17)	0.73	0.05 (0.13)	0.60	0.18 (0.23)
Claude Haiku 3.5	0.27	0.49 (0.43)	0.33	0.46 (0.41)	0.47	0.38 (0.45)	0.33	0.44 (0.42)	0.27	0.58 (0.42)	0.27	0.49 (0.41)
Claude Sonnet 4	0.33	0.47 (0.45)	0.27	0.57 (0.43)	0.60	0.34 (0.47)	0.33	0.47 (0.46)	0.27	0.56 (0.41)	0.20	0.60 (0.41)
Grok 4	**0.87**	0.03 (0.09)	**0.73**	0.08 (0.13)	0.67	0.22 (0.33)	**0.73**	0.17 (0.30)	**1.0**	**0.0** (0.0)	**1.0**	**0.0** (0.0)
Gemini 2.5 Flash	0.73	0.16 (0.34)	0.53	0.14 (0.18)	**0.87**	0.07 (0.25)	0.47	0.32 (0.38)	0.93	**0.0** (0.01)	0.47	0.36 (0.42)
Llama 4 Scout 17B	0.13	0.79 (0.37)	0.0	0.77 (0.33)	0.13	0.79 (0.37)	0.0	0.77 (0.33)	0.20	0.77 (0.4)	0.0	0.81 (0.28)

Table D.2: **Counterfactual reasoning** results on CausalARC logical reasoning task `SCMdky5` (Figure D.3), where causal functions are a single logical operator. Scores are over five random prompt samples. Results are disaggregated by prompt formulation: L1 prompts include only L1 demonstrations (Figure D.4), while L3 prompts include alternating L1 and L3 demonstrations (Figure D.5). Both formulations feature a counterfactual test case. Metrics are accuracy (✓; values between 0 and 1; higher is better) and Hamming distance (HD, mean and standard deviation; values between 0 and 1; lower is better). Best performance per column is in bold.

	COMPOSITION				ALTERNATION			
	L1		L3		L1		L3	
	✓↑	*HD* ↓	✓↑	*HD* ↓	✓↑	*HD* ↓	✓↑	*HD* ↓
GPT-4o mini	0.27	0.34 (0.25)	0.27	0.34 (0.24)	0.13	0.24 (0.11)	0.13	0.24 (0.19)
o4-mini	0.47	0.18 (0.23)	0.40	0.19 (0.2)	0.40	**0.13** (0.11)	0.53	0.14 (0.21)
Claude Haiku 3.5	0.33	0.51 (0.42)	0.20	0.57 (0.36)	0.13	0.50 (0.39)	0.20	0.57 (0.40)
Claude Sonnet 4	0.33	0.45 (0.46)	0.07	0.65 (0.38)	0.13	0.50 (0.38)	0.13	0.51 (0.41)
Grok 4	**0.67**	**0.06** (0.13)	**0.93**	**0.02** (0.06)	**0.53**	0.19 (0.22)	**0.73**	**0.10** (0.19)
Gemini 2.5 Flash	0.27	0.48 (0.45)	0.27	0.58 (0.42)	0.40	0.38 (0.45)	0.07	0.69 (0.35)
Llama 4 Scout 17B	0.07	0.74 (0.39)	0.0	0.86 (0.28)	0.0	0.86 (0.29)	0.07	0.80 (0.34)

Table D.3: **Counterfactual reasoning** results on CausalARC logical reasoning tasks, where causal functions are multiple logical operators. Mixed operators include two cases: (1) task `SCMu3am`, where causal functions alternate by row of the input array (*or*, *and*); and (2) task `SCMtcbq`, where logical operators compose (*xor* after *and*). Scores are over five random prompt samples. Results are disaggregated by prompt formulation: L1 prompts include only L1 demonstrations (Figure D.4), while L3 prompts include alternating L1 and L3 demonstrations (Figure D.5). Both formulations feature a counterfactual test case. Metrics are accuracy (✓; higher is better) and Hamming distance (HD, mean and standard deviation; lower is better). Best performance per column is in bold.

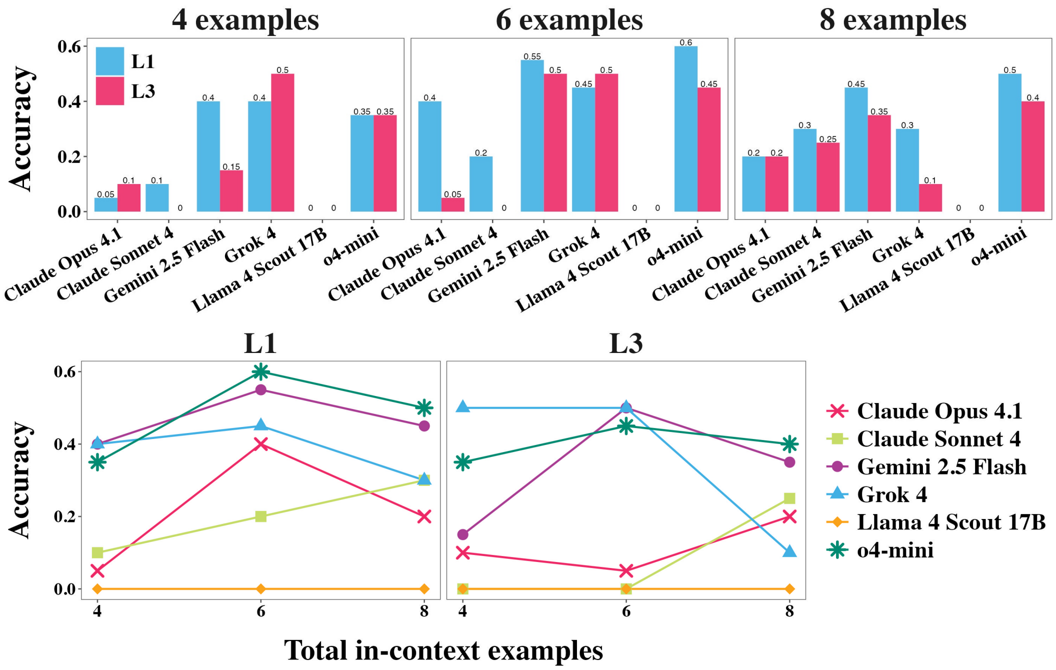

Figure D.16: **Program synthesis** accuracy by total in-context examples. For each of four tasks, five prompts were sampled per level of the PCH. Accuracy was measured with respect to the output array generated by passing a test input array to the LM-generated Python method. Tasks were sampled from `SCMm5ob` (counting), `SCMev5t` (counting), `SCMfwpq` (extension), and `SCMz750` (extension).

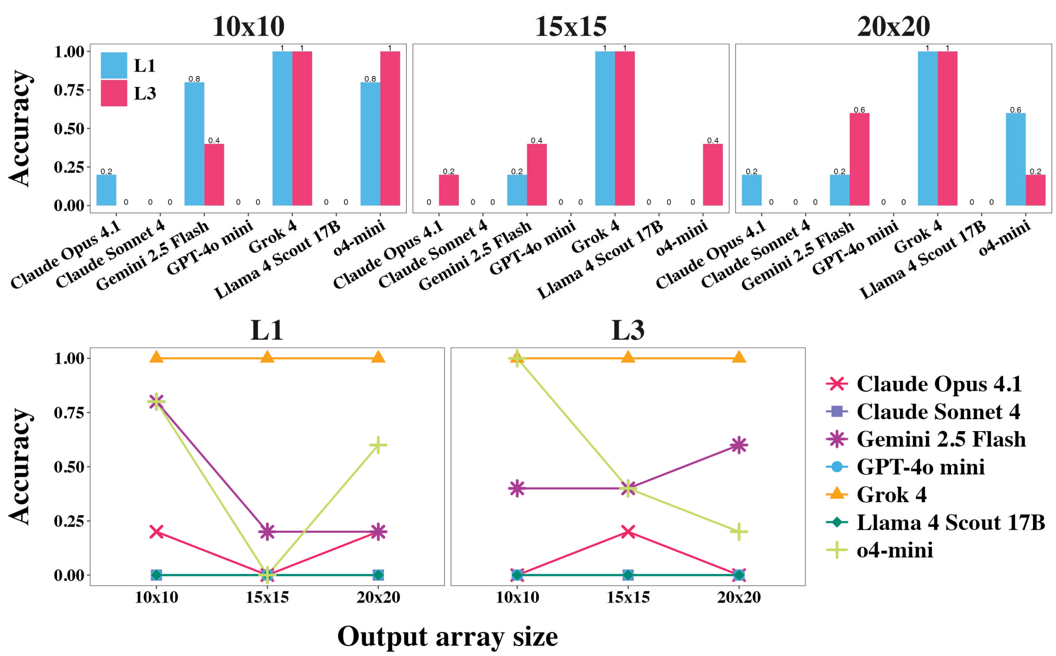

Figure D.17: **Causal discovery with logical reasoning** accuracy by scaling complexity with respect to output array size. For each of three CausalARC tasks sampled from task `SCMtcbq` (where logical operators compose *xor* after *and*), five prompts were sampled per level of the PCH. The LM was asked to predict the logical operators acting as causal functions, which requires knowledge of the causal parents of each output cell as a prerequisite.

n	LOGICAL AND	LOGICAL OR	LOGICAL XOR
1k	1.0	0.0	40.0
5k	1.0	0.0	42.0
10k	0.0	0.0	40.0

Table D.4: Causal discovery with PC algorithm [39] on CausalARC task `SCMdky5`. Distributions of L1 arrays were sampled at $n = 1000$, $n = 5000$, and $n = 10000$ observations per distribution. Arrays were flattened before being passed to PC. Reported values are the structural Hamming distance for the predicted causal graph with respect to the true graph as data sample size increases. A structural Hamming distance of 0 indicates that the output graph was identical to the true graph. PC algorithm was performed with chi-square conditional independence tests ($\alpha = 0.01$) using the `causal-learn` Python package (https://causal-learn.readthedocs.io/en/latest/).

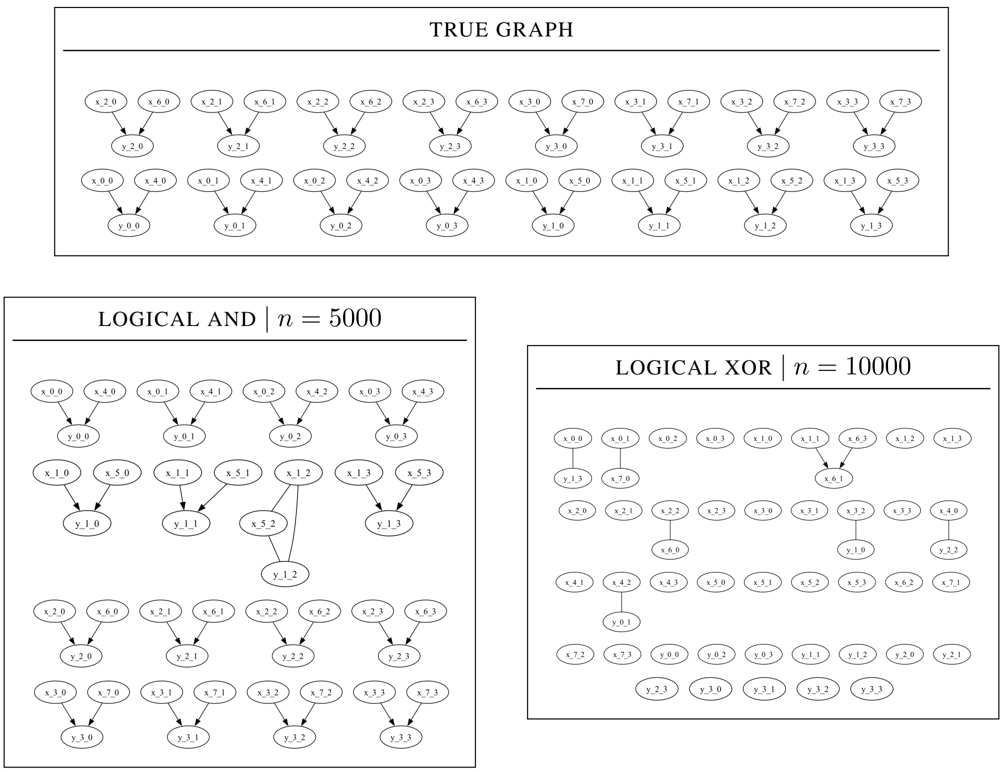

Figure D.18: Causal graphs predicted by PC algorithm [39] for CausalARC task `SCMdky5`. PC algorithm was performed with chi-square conditional independence tests ($\alpha = 0.01$) using the `causal-learn` Python package (https://causal-learn.readthedocs.io/en/latest/). Data sample size is denoted by n.