Can VLMs Reason Robustly? A Neuro-Symbolic Investigation

Anonymous authors
Paper under double-blind review

Abstract

Vision-Language Models (VLMs) have been applied to a wide range of reasoning tasks, yet it remains unclear whether they can reason robustly under distribution shifts. In this paper, we study covariate shifts in which the perceptual input distribution changes while the underlying prediction rules do not. To investigate this question, we consider visual deductive reasoning tasks, where a model is required to answer a query given an image and logical rules defined over the object concepts in the image. Empirically, we find that VLMs fine-tuned through gradient-based end-to-end training can achieve high in-distribution accuracy but fail to generalize under such shifts, suggesting that fine-tuning does not reliably induce the underlying reasoning function. This motivates a neuro-symbolic perspective that decouples perception from reasoning. However, we further observe that recent neuro-symbolic approaches that rely on black-box components for reasoning can still exhibit inconsistent robustness across tasks. To address this issue, we propose VLC, a neuro-symbolic method that combines VLM-based concept recognition with circuit-based symbolic reasoning. In particular, task rules are compiled into a symbolic program, specifically a circuit, which executes the rules exactly over the object concepts recognized by the VLM. Experiments on three visual deductive reasoning tasks with distinct rule sets show that VLC consistently achieves strong performance under covariate shifts, highlighting its ability to support robust reasoning.

1 Introduction

Vision-Language Models (VLMs) have recently been applied to a wide range of reasoning tasks, including abstract (Chollet, 2019; Unsal & Akkus, 2025), temporal (Li et al., 2024b; Fu et al., 2025), and document reasoning (Zhu et al., 2024; Wang et al., 2024), as well as relational, attributive, and order understanding (Yüksekgönül et al., 2023a; Zhao et al., 2022). Despite these advances, it remains unclear whether VLMs can reason robustly under distribution shifts. Here, we focus on *covariate shifts* (Shimodaira, 2000; Sugiyama et al., 2007; 2008; Bickel et al., 2009), where the distribution of the perceptual input changes, while the underlying rules for prediction do not. To study the robustness against covariate shifts, we use *visual deductive reasoning tasks*. In these tasks, an image containing multiple objects is provided, along with rules that define the reasoning function based on object concepts. The VLM is prompted to answer a query that requires reasoning over the object concepts using the given rules. See Fig. 1 (left) for an example. We focus on these tasks because they include samples with perceptual inputs of varying complexity and whose labels are generated by the same reasoning function. For instance, the reasoning function can be fixed as addition, while the perceptual input contains different digits (in shape and number), see Fig. 1.

End-to-end fine-tuning is the canonical approach for adapting VLMs to specific downstream tasks. Following this standard practice, we fine-tune VLMs on visual deductive reasoning tasks using the causal language modeling loss. As shown in Figure 1 (right), despite achieving high accuracy on in-distribution data, the fine-tuned models fail to generalize to out-of-distribution (OOD) data that involves a different number of objects but follows the same reasoning function. This result contrasts with the strong generalization often observed when VLMs are fine-tuned for other vision-language tasks like recognition (Li et al., 2022), and suggests that gradient-based fine-tuning does not necessarily enable the model to learn the underlying reasoning function.

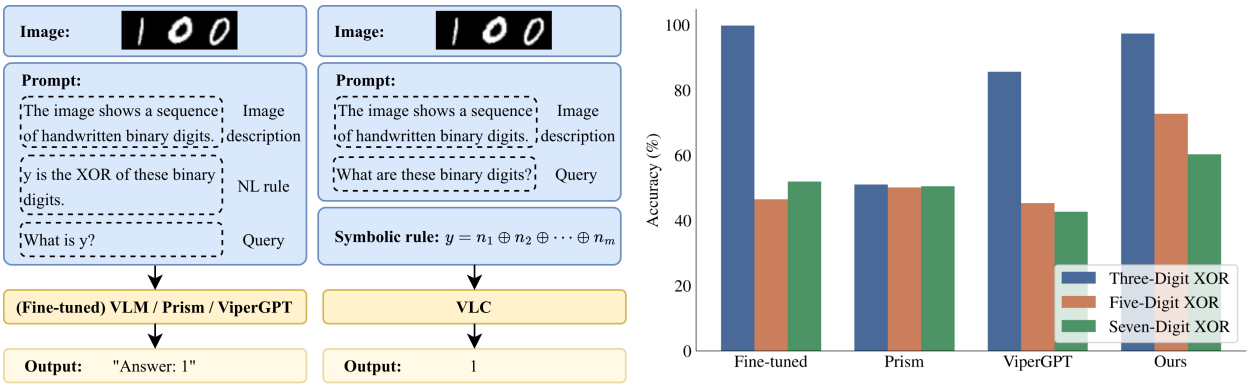


Figure 1: **Left:** Given an image containing multiple objects and natural-language rules about object concepts, the (fine-tuned) VLM, Prism, and ViperGPT are required to answer a query based on the image by reasoning with the given rules. **Middle:** VLC decouples perception from reasoning and infers the final answer by applying the symbolic rule compiled into the circuit. **Right:** Performance of different paradigms on datasets sharing the same reasoning function but differing in the number of objects per image. Here, the reasoning function is *logical XOR* (see Section 2 for the task definition), and the three datasets contain images with three, five, and seven handwritten binary digits, respectively. The models are required to output the XOR of these digits.

To see whether this limitation stems from end-to-end VLM reasoning, we evaluate two recent neuro-symbolic approaches that decouple perception from reasoning: Prism (Qiao et al., 2024) and ViperGPT (Surís et al., 2023). Prism delegates reasoning to a Large Language Model (LLM), whereas ViperGPT prompts an LLM to generate executable programs that call upon specialized pretrained models. However, as we shall see in Section 5.2, even with this decoupling of perception and reasoning, both approaches still show limited robustness under covariate shifts.

This leads to a natural question: *How can we encode the reasoning function into VLMs to achieve robust reasoning?* Inspired by recent studies (Cooper et al., 2025a; Al-Tahan et al., 2024) showing that VLMs excel at recognition tasks but struggle with reasoning, and in line with Qiao et al. (2024); Surís et al. (2023); Kamali & Kordjamshidi (2025), we propose VLC, a neuro-symbolic reasoning paradigm for VLMs that decomposes the end-to-end reasoning process into two sequential phases: *VLM-based concept recognition* and *circuit-based symbolic reasoning*. In the first phase, a VLM serves as the neural module, leveraging its strong recognition capabilities to identify object concepts in the input image. In the second phase, we use circuits (Oztok & Darwiche, 2014; Choi et al., 2020; Vergari et al., 2021) as the symbolic module, which compile the provided rules into its structure (Darwiche, 2011; Lagniez & Marquis, 2017; Muise et al., 2012). At inference time, the VLM first recognizes object concepts, after which the circuit applies the compiled rules to derive the final answer. By explicitly encoding the true reasoning function into the circuit, VLC ensures interpretable and robust reasoning and could allow for principled neuro-symbolic integration where constraints can be embedded in the learning pipeline of deep learning architecture (Manhaeve et al., 2018a; Ahmed et al., 2022; Chen et al., 2025a). For our purposes, and striving for simplicity, we adopt the two-stage prediction scenario of Chen et al. (2025a), where learning is decomposed into a first stage where the VLM is used to predict object concepts and then the circuit is used to make predictions, essentially firing the symbolic rules based on the observed concepts.

To evaluate different reasoning paradigms, we advocate going back to benchmarking simple visual deductive reasoning tasks where generalization can be controlled in a rigorous way. Specifically, we use the `rsbench` benchmark suite (Bortolotti et al., 2024) to generate datasets in which the perceptual input varies through the number of objects in the image, while labels are produced by the same reasoning function, *i.e.*, the covariate shift setting (Shimodaira, 2000; Sugiyama et al., 2007; 2008; Bickel et al., 2009). We train on samples with fewer objects and test on samples with more objects. In particular, we consider three distinct reasoning functions: *arithmetic addition*, *logical XOR*, and a *relational check*, respectively. Note that these datasets

differ from classic neuro-symbolic benchmarks, which often assume that each object is already perfectly segmented and provided as a separate image, and typically do not provide concept annotations or explicit rules specifying how object concepts map to the final label. Empirically, we find that even on these simple tasks, end-to-end fine-tuned VLMs fail to generalize to OOD samples, suggesting that end-to-end fine-tuning does not reliably learn the underlying reasoning function from data. In contrast, VLC consistently achieves competitive accuracy across all datasets under covariate shifts, indicating that a simple neuro-symbolic pipeline that explicitly encodes the reasoning function as an external symbolic program can in fact improve robustness. In addition, we conduct a series of ablation studies. Notably, we find that scaling up model size improves VLMs' performance on concept recognition but does not necessarily enhance their reasoning ability, which is consistent with findings in Cherti et al. (2023); Al-Tahan et al. (2024); Zhang et al. (2024a).

Our main contributions are: **(1)** In Section 3, we propose a neuro-symbolic method for visual deductive reasoning, VLC, that decomposes end-to-end reasoning into VLM-based concept recognition and circuit-based symbolic reasoning. **(2)** In Section 5, we empirically demonstrate that VLMs do not learn to emulate the underlying reasoning function through gradient-based fine-tuning. We further observe that fully-fledged neuro-symbolic approaches, Prism and ViperGPT, which rely on black-box components to complete reasoning, can be unreliable and yield inconsistent robustness across visual deductive reasoning tasks. **(3)** In Section 5, we demonstrate the consistently high robustness of VLC under covariate shifts across visual deductive reasoning tasks with distinct reasoning functions, highlighting the benefit of decoupling perception from reasoning and compiling the reasoning function into a symbolic program.

2 Visual Deductive Reasoning

Visual deductive reasoning tasks are designed to test whether a model can deduce the answer to a query from objects present in an image and rules for prediction. A key feature of these tasks is that rules are explicitly provided as part of the input. They differ from classical neuro-symbolic benchmarks (Bortolotti et al., 2024; Manhaeve et al., 2018b; Vermeulen et al., 2023), where each object is often perfectly segmented and provided as a separate image, and the rule specifying the reasoning function is typically not given. Consequently, the model has to infer the reasoning function from data and encode it in the learned parameters. These tasks also differ from standard visual question answering (VQA) benchmarks (Yüksekgönül et al., 2023a; Chen et al., 2024a; Hudson & Manning, 2019), where the reasoning function is often sample-specific, highly abstract, and not provided explicitly, *e.g.*, a sample that contains a radiograph image and a query like "Which organ appears abnormal in this radiograph?". The distinctive feature of visual deductive reasoning tasks allows us to investigate a focused question in this paper: *Even when the rule is explicitly available, can a model capture and apply it reliably under covariate shifts, and what type of approach best supports this behavior?*

Each sample from the task consists of three input components. First, the image contains multiple objects. We construct splits where training and validation images contain fewer objects, while test images contain more objects, creating a controlled covariate shift. Second, the task provides symbolic rules that concisely and precisely specify how object concepts map to the label. As pretrained VLMs may not interpret symbolic rules reliably, equivalent natural-language descriptions are also given. Third, a textual query asks for the value of the label.

In particular, we consider the following visual deductive reasoning tasks with distinct reasoning functions.

MNAdd. Each image contains two rows of handwritten digits from the MNIST dataset (LeCun et al., 2002), representing two multi-digit numbers. The task requires the model to compute the sum of these two numbers. The reasoning function is therefore the arithmetic addition, and the sum is used as the label y. Suppose that each number can be represented using m bits, denoted as $a_{m-1} \dots a_0$ and $b_{m-1} \dots b_0$, where a_{m-1} and b_{m-1} are the most significant bits. The reasoning function is formulated as the following symbolic rule:

$$\begin{aligned} x_i &= (a_i \wedge \neg b_i) \vee (\neg a_i \wedge b_i), \\ s_i &= (x_i \wedge \neg c_i) \vee (\neg x_i \wedge c_i), \\ c_{i+1} &= (a_i \wedge b_i) \vee (x_i \wedge c_i), \end{aligned} \tag{1}$$

where $i \in \{0, \ldots, m-1\}$ and $c_0 = 0$. Here, x_i denotes the XOR of a_i and b_i, s_i denotes the resulting sum bit, and c_{i+1} denotes the carry bit to the next position. The output is an $(m+1)$-bit binary number $c_m s_{m-1} \ldots s_0$, which is then converted to a decimal value y. The equivalent natural-language description is: *"Image description: The image shows two rows of handwritten digits. Each row represents a multi-digit number.\nRule: y is the sum of these two numbers.\nQuery: What is y?"*

MNLogic. Each image contains a sequence of handwritten binary digits from the MNIST dataset. The task requires the model to compute the XOR of these digits. Suppose m binary digits are present in an image, denoted as $n_1, \ldots, n_m$. The reasoning function, logical XOR, is formulated as the following symbolic rule:

$$\begin{aligned} z_1 &= (n_1 \wedge \neg n_2) \vee (\neg n_1 \wedge n_2), \\ z_i &= (z_{i-1} \wedge \neg n_{i+1}) \vee (\neg z_{i-1} \wedge n_{i+1}), \quad i \in \{2, \ldots, m-1\}, \\ y &= z_{m-1}, \end{aligned} \tag{2}$$

which corresponds to $y = n_1 \oplus n_2 \oplus \cdots \oplus n_m$. The equivalent natural-language description would be: *"Image description: The image shows a sequence of handwritten binary digits.\nRule: y is the XOR of these binary digits.\nQuery: What is y?"*

KandLogic. Each image contains multiple geometric primitives with various shapes and colors. The task requires the model to determine whether all objects of the same shape have the same color. The reasoning function is therefore a relational check. Suppose there are m geometric primitives in an image. Let the shape and color of the i-th primitive be denoted by s_i and c_i, respectively, for $i \in 1, \ldots, m$. The reasoning function is formulated as the following symbolic rule:

$$y = \bigwedge_{1 \le i < j \le m} \left(\mathbb{I}(s_i \neq s_j) \vee \mathbb{I}(c_i = c_j) \right), \tag{3}$$

where $\mathbb{I}(s_i \neq s_j)$ indicates whether two primitives have different shapes and $\mathbb{I}(c_i = c_j)$ indicates whether they have the same color. This rule is equivalent to $\forall i \neq j,\ \mathbb{I}(s_i = s_j) \rightarrow \mathbb{I}(c_i = c_j)$. The corresponding natural-language description is: *"Image description: The image shows multiple geometric primitives, each with a specific shape and color.\nRule: If all primitives with the same shape have the same color, then y is True. Otherwise, y is False.\nQuery: What is y?"*

3 VLC: Decoupling Perception and Reasoning

Deep neural networks excel at perception. They can extract statistical patterns from raw data, but they often struggle with structured, rule-based reasoning. In contrast, symbolic programs such as circuits excel at reasoning. They can enforce a given logical rule exactly, but they are less flexible when operating on raw, unstructured inputs (Manhaeve et al., 2018a; Ahmed et al., 2022; Chen et al., 2025a). Therefore, we advocate a simple design for visual deductive reasoning that combines the strengths of both. Instead of relying on a single deep model to perform end-to-end reasoning, we decompose the problem into perception and reasoning. Specifically, we introduce a neuro-symbolic reasoning paradigm, VLC, which uses a VLM for concept recognition and a circuit-based symbolic module to encode the rules (see Figure 2). Note that this is the simplest neuro-symbolic integration possible, where rules are still explicit (Chen et al., 2025a). While other more sophisticated pipelines and architectures are possible (Manhaeve et al., 2018a; Ahmed et al., 2022; Calanzone et al., 2025; De Smet et al., 2023; Kurscheidt et al., 2025; Lazzari et al., 2026), as we will show in our experiments, this simple strategy suffices when there is supervision over object concepts (Marconato et al., 2023; 2024; 2025).

3.1 Phase I: VLM-based Concept Recognition

Leveraging recent advances in visual recognition capabilities of VLMs (Cooper et al., 2025a; Al-Tahan et al., 2024), we directly prompt a VLM to identify object concepts in an image. For each task, we design a specific prompt instructing the VLM to recognize relevant object concepts (*e.g.*, digits, colors, shapes).

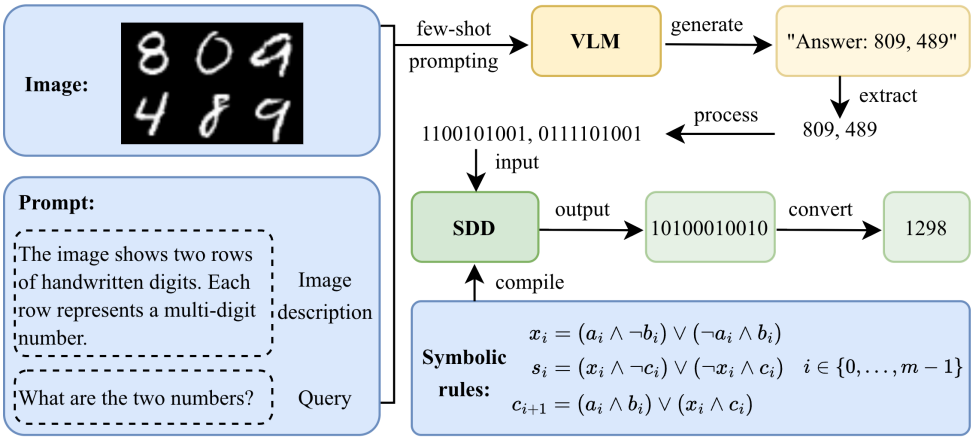


Figure 2: VLC consists of two phases: VLM-based concept recognition (yellow blocks) and circuit-based symbolic reasoning (green blocks). We use the `pySDD` compiler to compile the symbolic rules into the circuit, SDD in particular. During inference, the VLM is prompted to recognize object concepts in the input image. The generated response is then extracted and processed as the binary inputs to the SDD. The SDD uses the compiled rules to execute exact inference over the binary inputs and the output binary values are converted to the final answer.

Taking this prompt and the image as input, the VLM then generates recognition results. To further improve output consistency and facilitate result extraction, we employ in-context learning by incorporating few-shot examples in the prompt. Prompts for each task are provided in Appendix B. For instance, on the MNAdd task, the VLM takes as input the few-shot examples, an image, and a prompt—*"Image description: The image shows two rows of handwritten digits. Each row represents a multi-digit number.\nQuery: What are the two numbers?"*, and generates a response such as *"Answer: 640, 280"*.

3.2 Phase II: Circuit-based Symbolic Reasoning

Symbolic Programs. In Phase II, we execute the provided rules using a symbolic program rather than relying on a VLM to carry out end-to-end reasoning. A symbolic program is an explicit, executable representation of a rule-based computation, such as a logical formula, a set of constraints, or a small program in a domain-specific language. Given discrete inputs, a symbolic program computes the label by applying the rule exactly. This differs from neural networks, which typically compute outputs from inputs via learned approximations rather than exact rule execution.

Circuits. Among many instances of symbolic programs, we focus on Boolean circuits, which are computationally universal and can express any binary function over discrete variables (Arora & Barak, 2009). A Boolean circuit is a directed acyclic graph composed of logical operators (*e.g.*, $\wedge$, $\vee$, and $\neg$) that computes an output from binary inputs. While in knowledge compilation, one often considers circuit classes with certain structural properties, such as decomposability (Darwiche, 2001a), determinism (Darwiche, 2001b), and smoothness (Darwiche, 2001b), to enable tractable logical inference tasks like weighted model counting (Chavira & Darwiche, 2008), our setting does not strictly rely on them. In our setting, given a discrete assignment to the input variables, the circuit deterministically evaluates the rule and outputs its binary value, *i.e.*, whether the assignment satisfies the rule. This value is then used to derive the answer to the query. We obtain such circuits through knowledge compilation (Darwiche & Marquis, 2002; Chavira & Darwiche, 2005; Darwiche, 2002; Pipatsrisawat & Darwiche, 2008), which converts a symbolic specification of the rule (*e.g.*, a logical formula encoding the reasoning function) into an equivalent graphical structure that supports efficient inference. The circuit representation mechanizes rule execution and can be run on GPUs easily. In our experiments, we compile task-specific rules into circuits using the `pySDD` compiler (Meert & Choi, 2017), which produces sentential decision diagrams (SDDs) (Darwiche, 2011), a class of Boolean cir-

cuits in negation normal form (NNF)[1]. Our framework is not tied to SDDs, and other compilers (*e.g.*, those producing d-DNNF circuits) could be substituted without changing the overall procedure.

Unlike many neuro-symbolic approaches (Ahmed et al., 2022; 2023; Chen et al., 2025b; Calanzone et al., 2025), where learning happens by fine-tuning the whole system end-to-end, *i.e.*, by backpropagating through symbolic solvers and circuits (van Krieken et al., 2025) to improve task performance, we opt for the simpler solution of using the circuits as a deterministic mapper in a two-stage architecture in which there is no global fine-tuning. We show empirically that this suffices to achieve performance that is competitive or better than non-neuro-symbolic VLM baselines. Furthermore, we note that we could have used other symbolic formalisms to represent constraints (Michailidis et al., 2025). However, circuits have the benefit to have been extensively used for neuro-symbolic integration, and therefore there is a rich literature and software infrastructure we can exploit (Lazzari et al., 2026).

Symbolic Reasoning. Here, we elaborate on how the circuit performs symbolic reasoning over the object concepts recognized by the VLM. Assume we have compiled the task-specific rules into a circuit. As shown in Figure 2, in Phase I, the VLM is prompted to recognize object concepts from the input image. Thanks to its in-context learning ability, the VLM produces responses that follow the answer format shown in the few-shot examples. This makes it feasible to extract the recognized concepts from the textual response and convert them into binary values that serve as inputs to the circuit. Given this assignment, the circuit deterministically evaluates the rule and produces binary outputs, which we then convert into the final answer to the query.

We illustrate this procedure on the MNAdd task. Phase I produces a response such as "Answer: 640, 280". We parse the two recognized numbers and convert them into their binary representations, 1010000000 and 0100011000, yielding the corresponding input bits to the circuit. This circuit then evaluates the compiled addition rules to produce the output bits, 1110011000, which is converted back to a decimal value 920 as the predicted sum.

4 Related Work

Visual Deductive Reasoning Tasks. Natural-language deductive reasoning tasks have been widely studied. A variety of benchmarks (Han et al., 2024a;b; Parmar et al., 2024; Tafjord et al., 2021; Tian et al., 2021; Kassner et al., 2021; Saparov et al., 2023) have been established for these tasks, covering diverse types of rules and reasoning patterns, and are used to evaluate the deductive reasoning capabilities of LLMs. Several works (Chen et al., 2024b; Li et al., 2024a; Poesia et al., 2024; Zhang et al., 2023) have further leveraged these benchmarks to investigate specific research questions. In particular, Zhang et al. (2023) use these tasks to examine whether a language model can be trained end-to-end to learn the underlying reasoning function.

However, these deductive reasoning tasks are primarily language-based, with facts, rules, and queries all provided in text. Deductive reasoning tasks with visual inputs are comparatively underexplored, leaving the deductive reasoning capabilities of VLMs less well studied. Existing reasoning benchmarks for VLMs, including those adopted in neuro-symbolic approaches such as Prism and ViperGPT, primarily evaluate performance on general multimodal understanding and reasoning (Chen et al., 2024a), commonsense reasoning (Surís et al., 2023; Gupta & Kembhavi, 2023), abstract reasoning (Zhang et al., 2024b; Hersche et al., 2024; Yang et al., 2023b), relational or order understanding (Yüksekgönül et al., 2023a; Al-Tahan et al., 2024; Hudson & Manning, 2019), and temporal reasoning (Xiao et al., 2021). The reasoning functions in these tasks tend to be implicit and abstract. In contrast, visual deductive reasoning tasks feature reasoning functions that are explicitly formulated and provided as rules in the input. This explicitness makes it easier to generate datasets with the same reasoning function but varying patterns, providing an effective testbed to study whether a VLM can genuinely learn a reasoning function and thereby achieve robust reasoning.

VLM-based Paradigms. Several works (Qiao et al., 2024; Cooper et al., 2025b; Surís et al., 2023; Gupta & Kembhavi, 2023) decompose the end-to-end reasoning process of VLMs to enhance their reasoning capabilities. Specifically, Qiao et al. (2024); Cooper et al. (2025b) decompose the process into two phases—a VLM-

[1]In NNF circuits, NOT gates are restricted to appearing only on input variables.

based concept recognition phase and an LLM-based reasoning phase—and demonstrate that incorporating an LLM can augment the VLM's external knowledge and reasoning ability. Despite these improvements, the LLM remains a black box, and it is unclear whether it truly encodes a reasoning function. Surís et al. (2023); Gupta & Kembhavi (2023); Kamali & Kordjamshidi (2025) take a different approach by prompting an LLM to generate a program that decomposes the query into a sequence of steps, which are executed by calling various pretrained models, such as VLMs, to complete each step. These paradigms benefit from introducing intermediate subqueries that are easier to solve; however, they depend heavily on the quality of the generated program, and the reliance on multiple black-box pretrained models increases the risk of propagating errors through incorrect intermediate results.

Beyond reasoning, VLMs have also been used to develop annotation-free concept bottleneck models (CBMs). Traditional CBMs (Koh et al., 2020) consist of a concept recognition model, typically a CNN, and a sparse linear layer. The concept recognition model outputs scores for different concepts in the input image, while the sparse linear layer predicts the final class label based on the concept scores, making the predictions interpretable in terms of the recognized concepts. However, training the concept recognition model requires concept-level annotations, which are expensive to obtain. Recent approaches (Oikarinen et al., 2023; Yüksekgönül et al., 2023b; Yang et al., 2023a; Debole et al., 2025) eliminate this need by replacing the CNN with a VLM, resulting in annotation-free CBMs. Given a predefined concept vocabulary, with each concept paired with a textual description, the VLM is typically used in one of two ways: either directly prompted to identify whether a concept is present in the image, or used as a feature encoder to generate embeddings for both the image and the concepts, with their cosine similarity serving as concept scores. Overall, in these approaches, the VLM functions as a concept extractor for the downstream classification task.

5 Experiments

5.1 Experimental Setup

Datasets. We perform evaluations on three visual deductive reasoning tasks as described in Section 2. To test robustness under controlled covariate shifts, we use `rsbench` (Bortolotti et al., 2024) to generate, for each task, three datasets that differ only in the number of objects per image. Concretely, we vary this factor from 3 to 5 to 7, yielding the following dataset variants: *MNAdd-3dgt*, *MNAdd-5dgt*, *MNAdd-7dgt*; *MNLogic-3dgt*, *MNLogic-5dgt*, *MNLogic-7dgt*; and *KandLogic-3obj*, *KandLogic-5obj*, *KandLogic-7obj*. Examples of samples are provided in Appendix D.

Baselines. We compare VLC against the following reasoning paradigms. **End2end reasoning**: It refers to the original end-to-end reasoning process of pretrained VLMs. **End2end fine-tuning**: It refers to fine-tuning the pretrained VLMs on datasets with the fewest objects, *i.e.*, MNAdd-3dgt, MNLogic-3dgt, and KandLogic-3obj, respectively. **Prism** (Qiao et al., 2024): Similar to our method, Prism adopts a compositional reasoning paradigm which comprises a concept recognition phase and a reasoning phase. Unlike ours, Prism uses an LLM instead of a symbolic module for the reasoning phase. **ViperGPT** (Surís et al., 2023): ViperGPT prompts a code generation model to produce a program that breaks down a query into sequential steps. A Python interpreter then executes the program by calling different pretrained models, such as detection models or LLMs, to complete each step.

Models. We adopt the VLM, specifically Qwen2.5-VL-7B (Team, 2025), in both our methods and all baselines. In addition, for Prism, we adopt Qwen2.5-7B (Team, 2024) as the LLM; for ViperGPT, we employ GPT-4o (Achiam et al., 2023) as the code generation model and GroundingDINO (Liu et al., 2024) as the detection model. More implementation details are provided in Appendix C.

5.2 Main Results

Performance Comparison with Baselines. Table 1 reports the task accuracy of different reasoning paradigms on the MNAdd, MNLogic, and KandLogic tasks under covariate shifts.

Table 1: Performance comparison of different reasoning paradigms on the MNAdd, MNLogic, and KandLogic tasks under covariate shifts. All paradigms adopt a 7B VLM. Results are averaged over 5 random seeds (mean ± standard deviation). The best results are highlighted in bold, and the second-best results are underlined.

Task	**MNAdd**			**MNLogic**			**KandLogic**		
Paradigm	**3dgt**	**5dgt**	**7dgt**	**3dgt**	**5dgt**	**7dgt**	**3obj**	**5obj**	**7obj**
End2end RS	26.99 ± 0.13	6.95 ± 0.09	1.85 ± 0.02	71.73 ± 0.16	49.51 ± 0.24	50.05 ± 0.02	65.05 ± 0.09	62.11 ± 0.14	66.60 ± 0.10
End2end FT	**79.65 ± 0.04**	2.53 ± 0.00	1.35 ± 0.02	**99.83 ± 0.00**	46.55 ± 0.09	51.95 ± 0.13	**99.97 ± 0.00**	64.28 ± 0.02	57.76 ± 0.03
Prism	50.22 ± 0.13	46.21 ± 0.19	30.69 ± 0.13	51.06 ± 0.18	50.19 ± 0.40	50.53 ± 0.43	58.30 ± 0.10	55.06 ± 0.11	56.35 ± 0.10
ViperGPT	19.71 ± 0.04	1.92 ± 0.04	0.04 ± 0.02	85.66 ± 0.11	45.36 ± 0.17	42.74 ± 0.16	96.56 ± 0.08	**91.18 ± 0.09**	84.70 ± 0.09
VLC	54.62 ± 0.06	**57.79 ± 0.09**	**52.06 ± 0.13**	97.37 ± 0.04	**72.78 ± 0.14**	**60.31 ± 0.27**	95.97 ± 0.07	89.97 ± 0.17	**92.21 ± 0.12**

Compared to using pretrained VLMs for end-to-end reasoning, fine-tuning on datasets with the fewest objects (*i.e.*, the 3dgt/obj datasets) significantly improves the in-distribution performance, even achieving near-perfect accuracy on MNLogic and KandLogic tasks. However, these fine-tuned models fail to generalize since their task accuracy on datasets with more objects (*i.e.*, the 5dgt/obj and 7dgt-obj datasets) remains similar to or worse than that of the pretrained models. This suggests that end-to-end fine-tuning may only learn some statistical features from the training data, and thus does not ensure that the VLMs learn the correct reasoning function.

Prism adopts a compositional paradigm, with the reasoning phase delegated to a 7B LLM. We observe that Prism effectively improves performance over end-to-end reasoning on the MNAdd task while degrading accuracy on the MNLogic and KandLogic tasks. These results indicate that the LLM captures the arithmetic reasoning function but struggles with logical reasoning functions. Therefore, delegating reasoning to a black-box LLM cannot guarantee high performance across reasoning tasks, as it depends on the LLM's inherently unknown reasoning capabilities.

ViperGPT generates programs using a code-generation model and executes them by calling various pretrained models. We observe that it attains low accuracy on MNAdd, while demonstrating high accuracy on KandLogic. Given that the generated programs are syntactically and logically correct, this discrepancy in accuracy is likely due to the varying performance of the underlying pretrained models. For instance, as shown in Appendix E, the detection model performs well on KandLogic but fails to detect objects accurately on MNAdd. This underscores a potential limitation of relying on multiple black-box models. Specifically, while the final performance can be high when these models produce correct results, errors at any step may propagate and substantially degrade overall performance.

Finally, we observe that VLC substantially improves the performance over end-to-end reasoning on all datasets across different tasks. On average, they increase accuracy from 54.59% to 82.65% on the 3dgt/obj dataset, from 39.52% to 73.51% on the 5dgt/obj dataset, and from 39.50% to 68.19% on the 7dgt/obj dataset. These consistent gains suggest that **structurally encoding the true reasoning function within a circuit and then incorporating this circuit within the overall architecture enable high-performing and robust reasoning under covariate shifts.**

Relationship with Concept Accuracy. For compositional reasoning paradigms, *i.e.*, Prism and VLC, we further examine the relationship between task accuracy and the accuracy of their VLMs in recognizing object concepts, referred to as *concept accuracy*. Since Prism and VLC adopt the same VLM with the same prompting strategy in their first (concept recognition) phase, they achieve very similar concept accuracy, as shown in Figure 3. Nevertheless, the task accuracy of Prism is consistently lower than its concept accuracy across datasets, indicating that its reasoning performance is constrained not only by the VLM's recognition capability but also by the LLM's reasoning ability. In contrast, the task accuracy of VLC closely matches its concept accuracy. This suggests that **the performance of VLC on reasoning tasks is dependent solely on the VLM's recognition capability, which is attributed to the fact that VLC structurally encodes the true reasoning function within the circuit.** Thus, the incorporation of a circuit effectively improves the reliability of the overall pipeline.

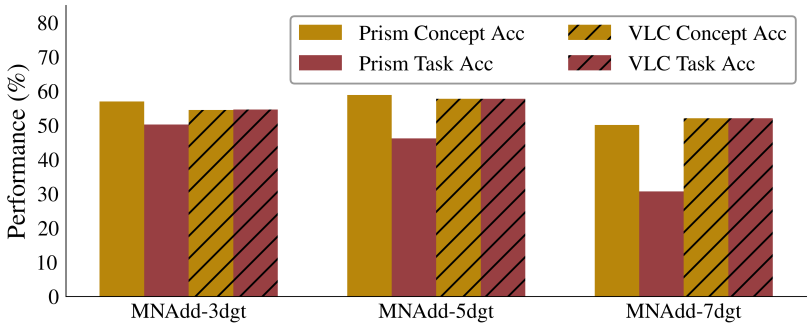


Figure 3: Comparison of concept accuracy and task accuracy for Prism and VLC across different datasets on the MNAdd task. While Prism and VLC achieve similar concept accuracy, VLC maintains a much smaller gap between concept accuracy and task accuracy, whereas this gap grows substantially for Prism as task complexity increases. This suggests that VLC ensures more robust reasoning under covariate shift.

Table 2: Concept and task accuracies of VLC on the MNAdd, MNLogic, and KandLogic tasks under covariate shifts. *"Before ft"* refers to VLC with a pretrained 7B VLM. *"After ft"* denotes VLC using a 7B VLM fine-tuned on the 3dgt/obj datasets for concept recognition. Results are averaged over 5 random seeds (mean ± standard deviation). The symbol ↑ means that the corresponding accuracy is improved after fine-tuning.

Task	**MNAdd**			**MNLogic**			**KandLogic**		
Acc	**3dgt**	**5dgt**	**7dgt**	**3dgt**	**5dgt**	**7dgt**	**3obj**	**5obj**	**7obj**
Concept Acc (before ft)	54.49 ± 0.05	57.76 ± 0.09	52.06 ± 0.13	97.10 ± 0.03	64.50 ± 0.13	48.76 ± 0.15	91.49 ± 0.06	72.97 ± 0.13	62.43 ± 0.31
Concept Acc (after ft)	86.46 ± 0.04 ↑	68.32 ± 0.05 ↑	53.07 ± 0.10 ↑	99.80 ± 0.00 ↑	88.39 ± 0.03 ↑	74.90 ± 0.08 ↑	99.97 ± 0.00 ↑	96.50 ± 0.11 ↑	85.75 ± 0.15 ↑
Task Acc (before ft)	54.62 ± 0.06	57.79 ± 0.09	52.06 ± 0.13	97.37 ± 0.04	72.78 ± 0.14	60.31 ± 0.27	95.97 ± 0.07	89.97 ± 0.17	92.21 ± 0.12
Task Acc (after ft)	86.46 ± 0.04 ↑	68.32 ± 0.05 ↑	53.07 ± 0.10 ↑	99.80 ± 0.00 ↑	90.54 ± 0.04 ↑	83.77 ± 0.06 ↑	100.00 ± 0.00 ↑	99.22 ± 0.05 ↑	97.37 ± 0.08 ↑

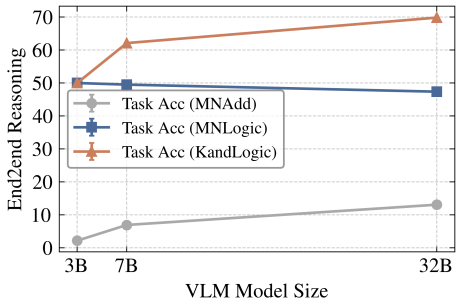
5.3 Ablation Studies

Effect of Fine-tuning on Concept Recognition. It has been shown in previous sections that fine-tuning VLMs on reasoning tasks does not guarantee high OOD performance. This raises a natural question: *What is the effect of fine-tuning on recognition tasks?* More specifically, *if we fine-tune VLMs to recognize object concepts in the training data, can the fine-tuned models achieve high recognition performance on OOD data?*

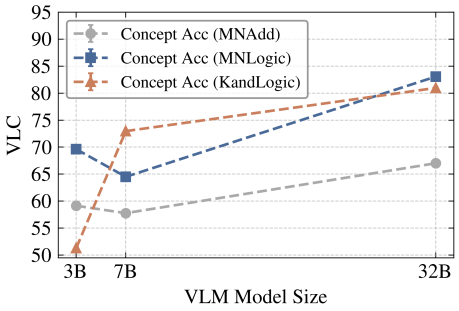
To answer this, we fine-tune VLMs on datasets with the fewest objects, where each sample consists of an image and the concept labels of all objects. We then replace the pretrained VLM in VLC with the fine-tuned one, and evaluate both the concept accuracy and the task accuracy of the resulting model.

As shown in Table 2, **unlike fine-tuning on reasoning tasks, fine-tuning on recognition tasks not only yields high in-distribution concept accuracy but also effectively improves concept accuracy on OOD datasets.** We hypothesize that the features used in concept recognition are relatively stable across distributions, compared with those used in reasoning. Thanks to this improvement, the task accuracy of VLC also increases across all datasets, as it is inherently dependent on the concept accuracy. It turns out that VLC with a fine-tuned VLM becomes the highest-performing model among all reasoning paradigms.

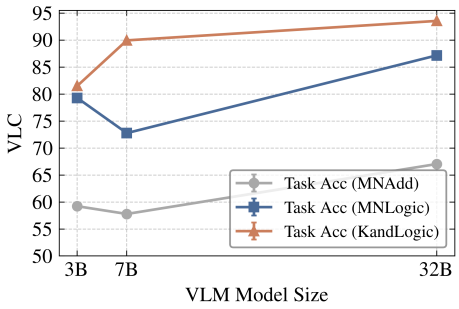
Effect of Scaling on Reasoning Capabilities of VLMs. Here, we aim to investigate the impact of model size scaling on the reasoning capabilities of VLMs. To this end, we vary the size of VLMs and evaluate their performance under the end-to-end reasoning paradigm. Figure 4a presents the task accuracies of this paradigm on the MNAdd-5dgt, MNLogic-5dgt, and KandLogic-5obj datasets. We observe that increasing the model size improves reasoning performance on MNAdd-5dgt and KandLogic-5obj, while failing to improve performance on MNLogic-5dgt, where the task accuracies remain around 50%. These results suggest that **scaling up the size of VLMs may not enhance reasoning for certain reasoning functions**, sharing similar views with Al-Tahan et al. (2024), which shows that scaling cannot aid relational reasoning.



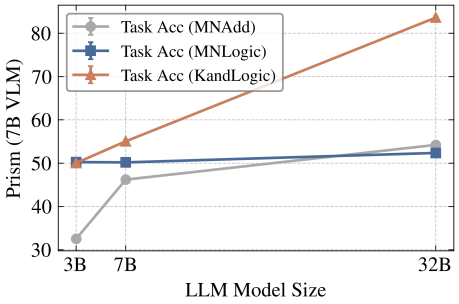
(a) Task Acc of End2end Reasoning



(b) Concept Acc of VLC



(c) Task Acc of VLC



(d) Task Acc of Prism (7B VLM)

Figure 4: (a) Effect of scaling up VLM size in End2end reasoning. (b-c) Effect of scaling up VLM size in VLC. (d) Effect of scaling up LLM size in Prism. Results are averaged over 5 random seeds, and error bars represent standard deviations.

Effect of Scaling on Recognition Capabilities of VLMs. To explore the impact of model size scaling on the recognition capabilities of VLMs, we apply VLC with varying VLM sizes and examine concept accuracies of the resulting models on the MNAdd-5dgt, MNLogic-5dgt, and KandLogic-5obj datasets. As shown in Figure 4b, an upward trend in concept accuracy is observed as the model size scales up, despite a slight dip from 3B to 7B. Specifically, when increasing from 3B to 32B, concept accuracies improve by 13%, 19%, and 58% on the three datasets, respectively. These results suggest that **scaling up the size of VLMs effectively enhances their abilities in concept recognition**, aligning with findings from Cherti et al. (2023); Al-Tahan et al. (2024); Zhang et al. (2024a). Besides, due to the compositional nature of VLC, task accuracies also gain improvement, as shown in Figure 4c.

Effect of Scaling on Reasoning Capabilities of LLMs. Here, we attempt to study the effect of model size scaling on the reasoning capabilities of LLMs. To this end, we apply Prism with a fixed 7B VLM and an LLM of varying size, and evaluate the resulting task accuracies on the MNAdd-5dgt, MNLogic-5dgt, and KandLogic-5obj datasets, as shown in Figure 4d. We find that increasing the LLM size improves reasoning performance on MNAdd-5dgt and KandLogic-5obj, but has little effect on MNLogic-5dgt, where the task accuracy improves by only 2% despite a 29B increase in LLM size. These results suggest that, similar to our findings on VLMs, **scaling up the size of LLMs may not enhance reasoning for certain reasoning functions**.

6 Conclusions

In this paper, we study the problem of robust VLM reasoning and present a neuro-symbolic investigation. We demonstrate that incorporating a symbolic program, a circuit specifically, with a VLM helps improve its performance and robustness on visual deductive reasoning tasks. In contrast, we find that end-to-end gradient-based fine-tuning and scaling up model size do not guarantee that VLMs capture the correct reasoning functions.

We also discuss several limitations and promising directions for future work. First, our current paradigm assumes access to symbolic rules, whereas in practice such rules are often expressed in natural language; extending the paradigm to automatically extract symbolic rules from natural-language descriptions would broaden its applicability. Second, the current symbolic module is task-specific, requiring a separate circuit for each reasoning function; developing a more flexible symbolic module that can accommodate multiple reasoning functions is a promising direction. Third, the overall performance of the proposed paradigm remains dependent on the VLM's recognition capability, suggesting the need for methods that reduce the sensitivity of symbolic reasoning to imperfect visual concept recognition. A more detailed discussion of these limitations and directions is provided in Appendix A.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Kareem Ahmed, Stefano Teso, Kai-Wei Chang, Guy Van den Broeck, and Antonio Vergari. Semantic probabilistic layers for neuro-symbolic learning. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, volume 35, pp. 29944–29959. Curran Associates, Inc., 2022.

Kareem Ahmed, Kai-Wei Chang, and Guy Van den Broeck. A pseudo-semantic loss for autoregressive models with logical constraints. In *NeuIPS*, 2023.

Haider Al-Tahan, Quentin Garrido, Randall Balestriero, Diane Bouchacourt, Caner Hazirbas, and Mark Ibrahim. Scaling vision-language models does not improve relational understanding: The right learning objective helps. In *CVPR*, pp. 8353–8358, 2024.

Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

Steffen Bickel, Michael Brückner, and Tobias Scheffer. Discriminative learning under covariate shift. *Journal of Machine Learning Research*, 10(9), 2009.

Samuele Bortolotti, Emanuele Marconato, Tommaso Carraro, Paolo Morettin, Emile van Krieken, Antonio Vergari, Stefano Teso, and Andrea Passerini. A neuro-symbolic benchmark suite for concept quality and reasoning shortcuts. In *NeurIPS*, 2024.

Diego Calanzone, Stefano Teso, and Antonio Vergari. Logically consistent language models via neuro-symbolic integration. In *The Thirteenth International Conference on Learning Representations*, 2025.

Mark Chavira and Adnan Darwiche. Compiling bayesian networks with local structure. In *IJCAI*, 2005.

Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artif. Intell.*, 172(6-7):772–799, 2008.

Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, and Feng Zhao. Are we on the right way for evaluating large vision-language models? In *NeurIPS*, 2024a.

Weixin Chen, Simon Yu, Huajie Shao, Lui Sha, and Han Zhao. Neural probabilistic circuits: Enabling compositional and interpretable predictions through logical reasoning. *arXiv preprint arXiv:2501.07021*, 2025a.

Weixin Chen, Simon Yu, Huajie Shao, Lui Sha, and Han Zhao. Neural probabilistic circuits: Enabling compositional and interpretable predictions through logical reasoning. *arXiv preprint arXiv:2501.07021*, 2025b.

Xinyun Chen, Ryan A. Chi, Xuezhi Wang, and Denny Zhou. Premise order matters in reasoning with large language models. In *ICML*, 2024b.

Mehdi Cherti, Romain Beaumont, Ross Wightman, Mitchell Wortsman, Gabriel Ilharco, Cade Gordon, Christoph Schuhmann, Ludwig Schmidt, and Jenia Jitsev. Reproducible scaling laws for contrastive language-image learning. In *CVPR*, pp. 2818–2829. IEEE, 2023.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic modeling. Technical report, University of California, Los Angeles (UCLA), 2020.

François Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

Avi Cooper, Keizo Kato, Chia-Hsien Shih, Hiroaki Yamane, Kasper Vinken, Kentaro Takemoto, Taro Sunagawa, Hao-Wei Yeh, Jin Yamanaka, Ian Mason, et al. Rethinking vlms and llms for image classification. *Scientific Reports*, 15(1):1–20, 2025a.

Avi Cooper, Keizo Kato, Chia-Hsien Shih, Hiroaki Yamane, Kasper Vinken, Kentaro Takemoto, Taro Sunagawa, Hao-Wei Yeh, Jin Yamanaka, Ian Mason, et al. Rethinking vlms and llms for image classification. *Scientific Reports*, 15(1):19692, 2025b.

Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, 2001a.

Adnan Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. *J. Appl. Non Class. Logics*, 11(1-2):11–34, 2001b.

Adnan Darwiche. A compiler for deterministic, decomposable negation normal form. In *AAAI*, 2002.

Adnan Darwiche. SDD: A new canonical representation of propositional knowledge bases. In *IJCAI*, pp. 819–826, 2011.

Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 2002.

Lennert De Smet, Pedro Zuidberg Dos Martires, Robin Manhaeve, Giuseppe Marra, Angelika Kimmig, and Luc De Readt. Neural probabilistic logic programming in discrete-continuous domains. In *Uncertainty in Artificial Intelligence*, pp. 529–538. PMLR, 2023.

Nicola Debole, Pietro Barbiero, Francesco Giannini, Andrea Passerini, Stefano Teso, and Emanuele Marconato. If concept bottlenecks are the question, are foundation models the answer? *arXiv preprint arXiv:2504.19774*, 2025.

Chaoyou Fu, Yuhan Dai, Yongdong Luo, Lei Li, Shuhuai Ren, Renrui Zhang, Zihan Wang, Chenyu Zhou, Yunhang Shen, Mengdan Zhang, Peixian Chen, Yanwei Li, Shaohui Lin, Sirui Zhao, Ke Li, Tong Xu, Xiawu Zheng, Enhong Chen, Caifeng Shan, Ran He, and Xing Sun. Video-mme: The first-ever comprehensive evaluation benchmark of multi-modal llms in video analysis. In *CVPR*, 2025.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *CVPR*, pp. 14953–14962. IEEE, 2023.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, Lucy Sun, Alexander Wardle-Solano, Hannah Szabó, Ekaterina Zubova, Matthew Burtell, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, Ansong Ni, Linyong Nan, Jungo Kasai, Tao Yu, Rui Zhang, Alexander R. Fabbri, Wojciech Kryscinski, Semih Yavuz, Ye Liu, Xi Victoria Lin, Shafiq Joty, Yingbo Zhou, Caiming Xiong, Rex Ying, Arman Cohan, and Dragomir Radev. FOLIO: natural language reasoning with first-order logic. In *EMNLP*, pp. 22017–22031, 2024a.

Simeng Han, Aaron Yu, Rui Shen, Zhenting Qi, Martin Riddell, Wenfei Zhou, Yujie Qiao, Yilun Zhao, Semih Yavuz, Ye Liu, Shafiq Joty, Yingbo Zhou, Caiming Xiong, Dragomir Radev, Rex Ying, and Arman Cohan. P-FOLIO: evaluating and improving logical reasoning with abundant human-written reasoning chains. In *EMNLP*, pp. 16553–16565, 2024b.

Michael Hersche, Giacomo Camposampiero, Roger Wattenhofer, Abu Sebastian, and Abbas Rahimi. Towards learning to reason: Comparing llms with neuro-symbolic on arithmetic relations in abstract reasoning. *arXiv preprint arXiv:2412.05586*, 2024.

Drew A. Hudson and Christopher D. Manning. GQA: A new dataset for real-world visual reasoning and compositional question answering. In *CVPR*, 2019.

Danial Kamali and Parisa Kordjamshidi. Neptune: A neuro-pythonic framework for tunable compositional reasoning on vision-language. In *NeuIPS 2025 Workshop on Space in Vision, Language, and Embodied AI*, 2025.

Nora Kassner, Oyvind Tafjord, Hinrich Schütze, and Peter Clark. Beliefbank: Adding memory to a pre-trained language model for a systematic notion of belief. In *EMNLP*, pp. 8849–8861. Association for Computational Linguistics, 2021.

Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5338–5348. PMLR, 2020.

Leander Kurscheidt, Paolo Morettin, Roberto Sebastiani, Andrea Passerini, and Antonio Vergari. A probabilistic neuro-symbolic layer for algebraic constraint satisfaction. In *The 41st Conference on Uncertainty in Artificial Intelligence*, 2025.

Jean-Marie Lagniez and Pierre Marquis. An improved decision-dnnf compiler. In *IJCAI*, 2017.

Nicolas Lazzari, Valentina Presutti, and Antonio Vergari. To neuro-symbolic classification and beyond by compiling description logic ontologies to probabilistic circuits. *arXiv preprint arXiv:2601.14894*, 2026.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 2002.

Jialian Li, Yipin Zhang, Wei Shen, Yuzi Yan, Jian Xie, and Dong Yan. Boosting deductive reasoning with step signals in rlhf. *arXiv preprint arXiv:2410.09528*, 2024a.

Junnan Li, Dongxu Li, Caiming Xiong, and Steven C. H. Hoi. BLIP: bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pp. 12888–12900. PMLR, 2022.

Kunchang Li, Yali Wang, Yinan He, Yizhuo Li, Yi Wang, Yi Liu, Zun Wang, Jilan Xu, Guo Chen, Ping Lou, Limin Wang, and Yu Qiao. Mvbench: A comprehensive multi-modal video understanding benchmark. In *CVPR*, 2024b.

Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, and Lei Zhang. Grounding DINO: marrying DINO with grounded pre-training for open-set object detection. In *ECCV*, volume 15105 of *Lecture Notes in Computer Science*, pp. 38–55. Springer, 2024.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *Advances in neural information processing systems*, 31, 2018a.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In *NeurIPS*, 2018b.

Emanuele Marconato, Stefano Teso, Antonio Vergari, and Andrea Passerini. Not all neuro-symbolic concepts are created equal: Analysis and mitigation of reasoning shortcuts. *Advances in Neural Information Processing Systems*, 36:72507–72539, 2023.

Emanuele Marconato, Samuele Bortolotti, Emile van Krieken, Antonio Vergari, Andrea Passerini, and Stefano Teso. Bears make neuro-symbolic models aware of their reasoning shortcuts. In *UAI*, 2024.

Emanuele Marconato, Samuele Bortolotti, Emile van Krieken, Paolo Morettin, Elena Umili, Antonio Vergari, Efthymia Tsamoura, Andrea Passerini, and Stefano Teso. Symbol grounding in neuro-symbolic ai: A gentle introduction to reasoning shortcuts. *arXiv preprint arXiv:2510.14538*, 2025.

Wannes Meert and Arthur Choi. Pysdd. In *Recent Trends in Knowledge Compilation*, number 17381 in Dagstuhl Reports, 2017.

Kostis Michailidis, Dimos Tsouros, and Tias Guns. Cp-bench: Evaluating large language models for constraint modelling. In *28th European Conference on Artificial Intelligence (ECAI)*, 2025.

Christian J. Muise, Sheila A. McIlraith, J. Christopher Beck, and Eric I. Hsu. Dsharp: Fast d-dnnf compilation with sharpsat. In *Advances in Artificial Intelligence - 25th Canadian Conference on Artificial Intelligence*, 2012.

Tuomas P. Oikarinen, Subhro Das, Lam M. Nguyen, and Tsui-Wei Weng. Label-free concept bottleneck models. In *ICLR*, 2023.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022.

Umut Oztok and Adnan Darwiche. On compiling CNF into decision-dnnf. In *Principles and Practice of Constraint Programming - 20th International Conference*, 2014.

Mihir Parmar, Nisarg Patel, Neeraj Varshney, Mutsumi Nakamura, Man Luo, Santosh Mashetty, Arindam Mitra, and Chitta Baral. Logicbench: Towards systematic evaluation of logical reasoning ability of large language models. In *ACL*, pp. 13679–13707, 2024.

Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In *AAAI*, 2008.

Gabriel Poesia, Kanishk Gandhi, Eric Zelikman, and Noah D. Goodman. Certified deductive reasoning with language models. *Trans. Mach. Learn. Res.*, 2024, 2024.

Yuxuan Qiao, Haodong Duan, Xinyu Fang, Junming Yang, Lin Chen, Songyang Zhang, Jiaqi Wang, Dahua Lin, and Kai Chen. Prism: A framework for decoupling and assessing the capabilities of vlms. In *NeurIPS*, 2024.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Abulhair Saparov, Richard Yuanzhe Pang, Vishakh Padmakumar, Nitish Joshi, Mehran Kazemi, Najoung Kim, and He He. Testing the general deductive reasoning capacity of large language models using OOD examples. In *NeurIPS*, 2023.

Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

Masashi Sugiyama, Matthias Krauledat, and Klaus-Robert Müller. Covariate shift adaptation by importance weighted cross validation. *Journal of Machine Learning Research*, 8(5), 2007.

Masashi Sugiyama, Taiji Suzuki, Shinichi Nakajima, Hisashi Kashima, Paul Von Bünau, and Motoaki Kawanabe. Direct importance estimation for covariate shift adaptation. *Annals of the Institute of Statistical Mathematics*, 60(4):699–746, 2008.

Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *ICCV*, pp. 11854–11864, 2023.

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. Proofwriter: Generating implications, proofs, and abductive statements over natural language. In *ACL/IJCNLP*, volume ACL/IJCNLP 2021 of *Findings of ACL*, pp. 3621–3634. Association for Computational Linguistics, 2021.

Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL `https://qwenlm.github.io/blog/qwen2.5/`.

Qwen Team. Qwen2.5-vl, January 2025. URL `https://qwenlm.github.io/blog/qwen2.5-vl/`.

Jidong Tian, Yitian Li, Wenqing Chen, Liqiang Xiao, Hao He, and Yaohui Jin. Diagnosing the first-order logical reasoning ability through logicnli. In *EMNLP*, pp. 3738–3747. Association for Computational Linguistics, 2021.

Mert Unsal and Aylin Akkus. Easyarc: Evaluating vision language models on true visual reasoning. *arXiv preprint arXiv:2506.11595*, 2025.

Emile van Krieken, Pasquale Minervini, Edoardo Ponti, and Antonio Vergari. Neurosymbolic diffusion models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Antonio Vergari, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck. A compositional atlas of tractable circuit operations for probabilistic inference. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, pp. 13189–13201. Curran Associates, Inc., 2021.

Arne Vermeulen, Robin Manhaeve, and Giuseppe Marra. An experimental overview of neural-symbolic systems. In *Inductive Logic Programming - 32nd International Conference, ILP*, 2023.

Weihan Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang, Lei Zhao, Xixuan Song, Jiazheng Xu, Keqin Chen, Bin Xu, Juanzi Li, Yuxiao Dong, Ming Ding, and Jie Tang. Cogvlm: Visual expert for pretrained language models. In *NeurIPS*, 2024.

Junbin Xiao, Xindi Shang, Angela Yao, and Tat-Seng Chua. Next-qa: Next phase of question-answering to explaining temporal actions. In *CVPR*, 2021.

Yuan Yang, Siheng Xiong, Ali Payani, Ehsan Shareghi, and Faramarz Fekri. Harnessing the power of large language models for natural language to first-order logic translation. In *ACL*, pp. 6942–6959. Association for Computational Linguistics, 2024.

Yue Yang, Artemis Panagopoulou, Shenghao Zhou, Daniel Jin, Chris Callison-Burch, and Mark Yatskar. Language in a bottle: Language model guided concept bottlenecks for interpretable image classification. In *CVPR*, pp. 19187–19197. IEEE, 2023a.

Zhengyuan Yang, Linjie Li, Kevin Lin, Jianfeng Wang, Chung-Ching Lin, Zicheng Liu, and Lijuan Wang. The dawn of lmms: Preliminary explorations with gpt-4v (ision). *arXiv preprint arXiv:2309.17421*, 9(1): 1, 2023b.

Mert Yüksekgönül, Federico Bianchi, Pratyusha Kalluri, Dan Jurafsky, and James Zou. When and why vision-language models behave like bags-of-words, and what to do about it? In *ICLR*, 2023a.

Mert Yüksekgönül, Maggie Wang, and James Zou. Post-hoc concept bottleneck models. In *ICLR*, 2023b.

Honghua Zhang, Liunian Harold Li, Tao Meng, Kai-Wei Chang, and Guy Van den Broeck. On the paradox of learning to reason from data. In *IJCAI*, pp. 3365–3373, 2023.

Jingyi Zhang, Jiaxing Huang, Sheng Jin, and Shijian Lu. Vision-language models for vision tasks: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 46(8):5625–5644, 2024a.

Yizhe Zhang, He Bai, Ruixiang Zhang, Jiatao Gu, Shuangfei Zhai, Josh M. Susskind, and Navdeep Jaitly. How far are we from intelligent visual deductive reasoning? In *COLM*, 2024b.

Tiancheng Zhao, Tianqi Zhang, Mingwei Zhu, Haozhan Shen, Kyusong Lee, Xiaopeng Lu, and Jianwei Yin. Vl-checklist: Evaluating pre-trained vision-language models with objects, attributes and relations. *arXiv preprint arXiv:2207.00221*, 2022.

Fengbin Zhu, Ziyang Liu, Xiang Yao Ng, Haohui Wu, Wenjie Wang, Fuli Feng, Chao Wang, Huanbo Luan, and Tat Seng Chua. Mmdocbench: Benchmarking large vision-language models for fine-grained visual document understanding. *arXiv preprint arXiv:2410.21311*, 2024.

A Discussions

In this section, we discuss the limitations of the proposed method and provide some potential solutions.

Extracting Symbolic Rules from Natural Language. In this paper, we assume access to symbolic rules for visual deductive reasoning tasks. In practice, however, rules are often expressed in natural language. This raises an important question: *How can symbolic rules be extracted from natural-language descriptions?* One approach is to train a language model for this purpose. For example, Yang et al. (2024) present LOGICLLAMA, a fine-tuned LLaMA-7B model that translates natural-language sentences into first-order logic rules. In a similar manner, by fine-tuning an LLM on paired natural-language descriptions and symbolic rules defined for visual deductive reasoning tasks, it becomes possible to automate the extraction of symbolic rules from natural language.

Improving Flexibility of the Symbolic Module. In this paper, a distinct SDD is constructed for each reasoning function. In other words, switching to a different reasoning function in the visual deductive reasoning task requires replacing the SDD in the symbolic module. This raises a question: *Is it possible to design a more flexible symbolic module that can accommodate multiple reasoning functions?* One promising direction is to design the symbolic module as an ensemble of SDDs, each corresponding to a specific reasoning function. Subsequently, reinforcement learning (RL) (Ouyang et al., 2022) can be employed to adaptively select the appropriate SDD according to the input. The selected SDD is then used to perform the reasoning phase.

Reducing Dependence on Recognition Capabilities of VLMs. As shown in our ablation studies, the reasoning performance of VLC is dependent on the recognition capabilities of the VLM. Specifically, strong concept recognition by the VLM leads to high performance of VLC on visual deductive reasoning tasks. Conversely, if the VLM fails to recognize certain object concepts, the reasoning performance of VLC is negatively impacted. To avoid such entanglement, *is it possible to reduce the dependence of* VLC *on the VLM's recognition capabilities?* This also remains an open challenge in the field of neuro-symbolic learning. Addressing this could significantly improve the reliability and robustness of the overall pipeline.

B Prompt Design

Figure 5 presents the prompts designed for the end-to-end reasoning paradigm on the MNAdd, MNLogic, and KandLogic tasks, respectively. We use them to prompt VLMs to generate the reasoning results.

Figure 6 presents the prompts designed for VLC on the MNAdd, MNLogic, and KandLogic tasks, respectively. We use them, in the first phase, to prompt VLMs to generate the concept recognition results.

Figure 7 presents the prompts designed for Prism on the MNAdd, MNLogic, and KandLogic tasks, respectively. We use the prompts shown in the top row, in the first phase, to prompt VLMs to generate the concept recognition results, while using the prompts shown in the bottom row, in the second phase, to prompt LLMs to generate the reasoning results.

For ViperGPT, we use the provided templates to prompt the code generation model, replacing the placeholder string "INSERT_QUERY_HERE" with the prompts shown in Figure 5.

C Implementation Details

Here, we elaborate on the implementation details for all baselines as well as our proposed method. All trainings and evaluations are conducted on a single NVIDIA A100-SXM4 GPU with 80 GB of memory. Specifically, each evaluation under the same setting is repeated five times using random seeds 0, 1, 2, 3, and 4.

MNAdd-3dgt	MNLogic-3dgt	KandLogic-3obj
## Image description: The image shows two rows of handwritten digits. Each row represents a 3-digit number. ## Rule: Y is the sum of these two numbers. ## Query: What is Y?	## Image description: The image shows 3 handwritten binary digits. ## Rule: Y is the XOR of these binary digits. ## Query: What is Y?	## Image description: The image shows 3 geometric primitives, each with a specific shape (square, triangle, circle) and color (red, blue, yellow). ## Rule: If all primitives with the same shape have the same color, then Y is True. Otherwise, Y is False. ## Query: What is Y?

Figure 5: Prompts designed for the end-to-end reasoning paradigm across different tasks. These prompts are used to prompt VLMs to generate the reasoning results.

MNAdd-3dgt	MNLogic-3dgt	KandLogic-3obj
## Image description: The image shows two rows of handwritten digits. Each row represents a 3-digit number. ## Query: What are the two numbers?	## Image description: The image shows 3 handwritten binary digits. ## Query: What are these binary digits?	## Image description: The image shows 3 geometric primitives, each with a specific shape (square, triangle, circle) and color (red, blue, yellow). ## Query: What are the shape and color of each primitive?

Figure 6: Prompts designed for VLC across different tasks. These prompts are used in the first phase to prompt VLMs to generate the concept recognition results.

MNAdd-3dgt	MNLogic-3dgt	KandLogic-3obj
## Image description: The image shows two rows of handwritten digits. Each row represents a 3-digit number. ## Query: What are the two numbers?	## Image description: The image shows 3 handwritten binary digits. ## Query: What are these binary digits?	## Image description: The image shows 3 geometric primitives, each with a specific shape (square, triangle, circle) and color (red, blue, yellow). ## Query: What are the shape and color of each primitive?
## Rule: Y is the sum of these two numbers. ## Query: What is Y? Please output the value of Y only. Do not provide explanations.	## Rule: Y is the XOR of these binary digits. ## Query: What is Y? Please output 'True' or 'False' only. Do not provide explanations.	## Rule: If all primitives with the same shape have the same color, then Y is True. Otherwise, Y is False. ## Query: What is Y? Please output 'True' or 'False' only. Do not provide explanations.

Figure 7: Prompts designed for Prism across different tasks. The prompts in the top row are used in the first phase to prompt VLMs to generate the concept recognition results; the prompts in the bottom row are used in the second phase to prompt LLMs to generate the reasoning results.

MNAdd-3dgt
Input variables: a00, a01, a02, a03, a04, a05, a06, a07, a08, a09, b00, b01, b02, b03, b04, b05, b06, b07, b08, b09 Constant variables: c00 = False Hidden variables: c01, c02, c03, c04, c05, c06, c07, c08, c09, x00, x01, x02, x03, x04, x05, x06, x07, x08, x09 Output variables: Y00, Y01, Y02, Y03, Y04, Y05, Y06, Y07, Y08, Y09, Y10 Formulas: x00 = (a00 AND NOTb00) OR (NOTa00 AND b00), Y00 = (x00 AND NOTc00) OR (NOTx00 AND c00), c01 = (a00 AND b00) OR (x00 AND c00), x01 = (a01 AND NOTb01) OR (NOTa01 AND b01), Y01 = (x01 AND NOTc01) OR (NOTx01 AND c01), c02 = (a01 AND b01) OR (x01 AND c01), x02 = (a02 AND NOTb02) OR (NOTa02 AND b02), Y02 = (x02 AND NOTc02) OR (NOTx02 AND c02), c03 = (a02 AND b02) OR (x02 AND c02), x03 = (a03 AND NOTb03) OR (NOTa03 AND b03), Y03 = (x03 AND NOTc03) OR (NOTx03 AND c03), c04 = (a03 AND b03) OR (x03 AND c03), x04 = (a04 AND NOTb04) OR (NOTa04 AND b04), Y04 = (x04 AND NOTc04) OR (NOTx04 AND c04), c05 = (a04 AND b04) OR (x04 AND c04), x05 = (a05 AND NOTb05) OR (NOTa05 AND b05), Y05 = (x05 AND NOTc05) OR (NOTx05 AND c05), c06 = (a05 AND b05) OR (x05 AND c05), x06 = (a06 AND NOTb06) OR (NOTa06 AND b06), Y06 = (x06 AND NOTc06) OR (NOTx06 AND c06), c07 = (a06 AND b06) OR (x06 AND c06), x07 = (a07 AND NOTb07) OR (NOTa07 AND b07), Y07 = (x07 AND NOTc07) OR (NOTx07 AND c07), c08 = (a07 AND b07) OR (x07 AND c07), x08 = (a08 AND NOTb08) OR (NOTa08 AND b08), Y08 = (x08 AND NOTc08) OR (NOTx08 AND c08), c09 = (a08 AND b08) OR (x08 AND c08), x09 = (a09 AND NOTb09) OR (NOTa09 AND b09), Y09 = (x09 AND NOTc09) OR (NOTx09 AND c09), Y10 = (a09 AND b09) OR (x09 AND c09)
MNLogic-3dgt
Input variables: n01, n02, n03 Constant variables: None Hidden variables: z01 Output variables: Y Formulas: z01 = (n01 AND NOTn02) OR (NOTn01 AND n02), Y = (z01 AND NOTn03) OR (NOTz01 AND n03)
KandLogic-3obj
Input variables: s1_ne_s2, c1_eq_c2, s1_ne_s3, c1_eq_c3, s2_ne_s3, c2_eq_c3 Constant variables: None Hidden variables: None Output variables: Y Formulas: Y = (s1_ne_s2 OR c1_eq_c2) AND (s1_ne_s3 OR c1_eq_c3) AND (s2_ne_s3 OR c2_eq_c3)

Figure 8: Symbolic rules from different visual deductive reasoning tasks.

End2end Reasoning. In the end2end reasoning paradigm, we select Qwen2.5-VL-7B-Instruct as the VLM. We prompt this model using the prompt shown in Figure 5 along with 5 few-shot examples. The batch size in the data loader is set to 128.

End2end Fine-tuning. In the end2end fine-tuning paradigm, we select Qwen2.5-VL-7B-Instruct as the pretrained VLM. We fine-tune the model on the training samples for 5 epochs with a batch size of 4, accumulating gradients over 8 steps. We employ the AdamW optimizer with a fixed learning rate of 2×10^{-4}, a constant schedule, a warmup ratio of 0.03, and gradient clipping at a maximum norm of 0.3. Evaluation on validation samples is conducted every 100 steps, and the best model, determined by minimum evaluation loss, is restored after training. After fine-tuning, we evaluate the fine-tuned model in the zero-shot setting using the prompt shown in Figure 5. The batch size in the data loader is set to 128.

Prism. In Prism, we select Qwen2.5-VL-7B-Instruct as the VLM and Qwen2.5-7B-Instruct as the LLM. We prompt the VLM using the prompt shown in Figure 7 (top row) along with 5 few-shot examples, and prompt the LLM using the prompt shown in Figure 7 (bottom row) in the zero-shot setting. The batch size in the data loader is set to 128.

ViperGPT. In ViperGPT, GPT-4o generates a program for each task based on the templates provided in Surís et al. (2023). Once generated, the same program is applied to all test samples for that task. The generated programs frequently invoke both a detection model and a VLM. Specifically, we use Grounding-DINO-Base as the detection model and Qwen2.5-VL-7B-Instruct as the VLM.

VLC. The batch size in the data loader of VLC is set to 128. In the first phase of VLC, we prompt Qwen2.5-VL-7B-Instruct using the prompt shown in Figure 6 along with 5 few-shot examples. In the second

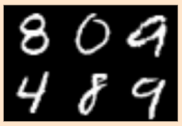
image	concept label	label
	[809, 489]	1298

image	concept label	label
	[1, 0, 0]	1

image	concept label	label
	[square, blue, triangle, red, circle, blue]	True

Figure 9: Samples from the MNAdd-3dgt (**top**), MNLogic-3dgt (**middle**), and KandLogic-3obj (**bottom**) datasets.

phase, the SDD follows the hierarchy of a randomly initialized vtree to compile the symbolic rules shown in Figure 8.

D Dataset Construction

In this section, we describe the construction of the datasets. Specifically, we use the `rsbench` benchmark to create datasets for various visual deductive reasoning tasks.

MNAdd. Datasets for the MNAdd task consist of images depicting two multi-digit numbers, with labels corresponding to the sum of these numbers. Taking the MNAdd-3dgt dataset as an instance, each image contains two rows, where each row comprises three horizontally arranged digit images randomly sampled from MNIST (LeCun et al., 2002). The label is the sum of the two three-digit numbers represented by the rows. We generate a total of 10000 training samples, 3000 test samples, and 2000 validation samples. Sample examples are provided in Figure 9 (top).

MNLogic. Datasets for the MNLogic task consist of images depicting a binary sequence, with labels corresponding to the XOR of the bits. For example, in the MNLogic-3dgt dataset, each image contains three horizontally arranged digit images, each being a 0 or 1, randomly sampled from MNIST (LeCun et al., 2002). The label is the XOR of the three bits represented in the image. We generate a total of 10000 training samples, 3000 test samples, and 2000 validation samples. Sample examples are provided in Figure 9 (middle).

KandLogic. Datasets for the KandLogic task consist of images depicting several geometric primitives, with labels indicating the result of a relational check between the shapes and colors of the primitives. Taking the KandLogic-3obj dataset as an example, each image contains three geometric primitives, whose shapes are randomly chosen from {circle, square, triangle} and colors randomly chosen from {red, yellow, blue}. The label indicates whether all primitives with the same shape have the same color. We generate a total of 10000 training samples, 3000 test samples, and 2000 validation samples. Sample examples are provided in Figure 9 (bottom).

E Performance Analysis of ViperGPT

Effect of the Detection Model. In the main experiments, we observe that ViperGPT (Surís et al., 2023) achieves competitive performance with the end-to-end reasoning paradigm on the MNLogic and KandLogic tasks, while exhibiting much lower task accuracy on the MNAdd task. Here, we investigate the cause of this inferior performance.

According to the program generated by GPT-4o (Yang et al., 2023b) for solving the MNAdd task (see Figure 14 (top row)), the first step in ViperGPT's pipeline is to invoke a detection model, GroundingDINO (Liu et al., 2024) in particular, to detect the digits in the input image. As shown in Figure 10, the detection model fails to correctly detect each digit, leading to errors that propagate through subsequent steps and ultimately result in incorrect answers. In contrast, as shown in Figures 11 and 12, the detection model successfully detects all digits or geometric primitives on the MNLogic and KandLogic tasks, which accounts for ViperGPT's high performance on these tasks.

These findings highlight that ViperGPT's reliance on the detection model enables high task accuracy when detection succeeds but also introduces the risk of cumulative errors that can degrade overall performance.

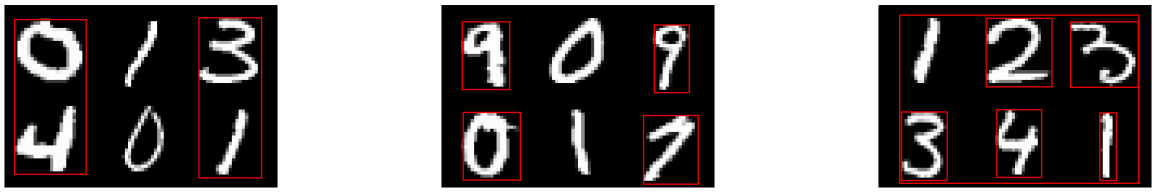


Figure 10: ViperGPT's detection results on three samples from the MNAdd-3dgt dataset. The detection model fails to detect each digit correctly.

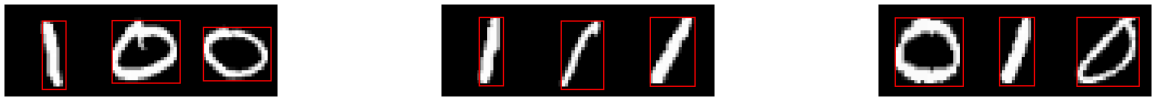


Figure 11: ViperGPT's detection results on three samples from the MNLogic-3dgt dataset. The detection model is able to detect each digit correctly.

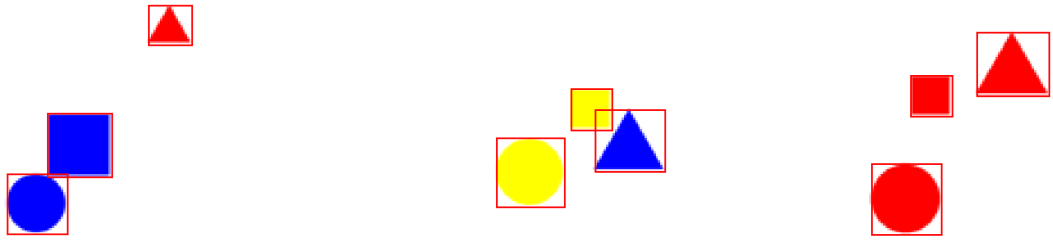


Figure 12: ViperGPT's detection results on three samples from the KandLogic-3obj dataset. The detection model is able to detect each geometric primitive correctly.

Effect of the Code Generation Model. In ViperGPT, the decomposition of the end-to-end reasoning process is primarily achieved through a program generated by a code generation model. Therefore, the quality of the generated program is of vital importance to the performance of ViperGPT. Here, we investigate the impact of the code generation model on ViperGPT's performance. In main experiments, we select GPT-4o (Yang et al., 2023b) as the code generation model. Figure 14 (top row) shows the programs generated by GPT-4o for different tasks. These programs are both syntactically and logically correct, thus enabling effective reasoning by breaking it down into a sequence of simpler steps.

We then choose CodeLlama-13B (Roziere et al., 2023) as the code generation model and examine the performance of ViperGPT. Figure 14 (bottom row) shows the programs generated for different tasks, among which those for the MNAdd and KandLogic tasks are logically incorrect. Thus, the reasoning process is decomposed into incorrect steps. Consequently, as shown in Figure 13, we observe a significant drop in task accuracy for these tasks. In particular, the incorrect program results in almost 0% accuracy on the MNAdd task.

These findings highlight the importance of selecting a code generation model that produces syntactically and logically correct programs, as errors in this step can critically undermine the overall reasoning process.

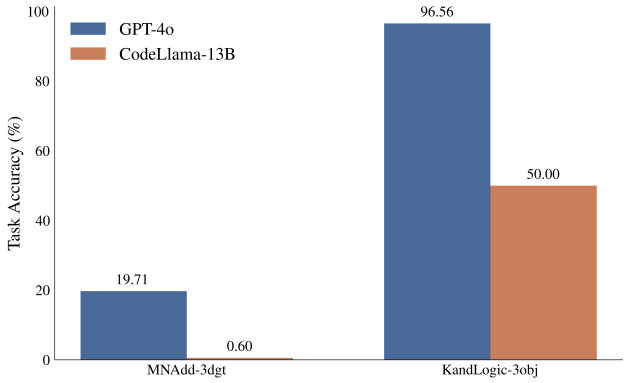


Figure 13: Comparison of ViperGPT's task accuracies with programs generated by GPT-4o and CodeLlama-13B.

MNAdd-3dgt

GPT-4o

```
def execute_command(image):
    image_patch = ImagePatch(image)

    # Find all digits in the image
    digit_patches = image_patch.find("digit")

    # Sort top 3 digits (smaller vertical_center), then left to right
    top_digits = sorted(
        [p for p in digit_patches if p.vertical_center < image_patch.vertical_center],
        key=lambda p: p.horizontal_center
    )

    # Sort bottom 3 digits (larger vertical_center), then left to right
    bottom_digits = sorted(
        [p for p in digit_patches if p.vertical_center >= image_patch.vertical_center],
        key=lambda p: p.horizontal_center
    )

    # Read digits
    top_number = int("".join(p.simple_query("What digit is this?") for p in top_digits))
    bottom_number = int("".join(p.simple_query("What digit is this?") for p in bottom_digits))

    # Compute sum
    return top_number + bottom_number
```

✓

CodeLlama-13B

```
def execute_command(image):
    image_patch = ImagePatch(image)
    digit_patches = image_patch.find("digit")
    digit_patches.sort(key=lambda x: x.vertical_center)
    top_digit_patches = digit_patches[:3]
    bottom_digit_patches = digit_patches[3:]
    top_digit_patches.sort(key=lambda x: x.horizontal_center)
    bottom_digit_patches.sort(key=lambda x: x.horizontal_center)
    top_digit_values = [int(digit_patch.simple_query("What is the value of this digit?")) for
digit_patch in top_digit_patches]
    bottom_digit_values = [int(digit_patch.simple_query("What is the value of this digit?")) for
digit_patch in bottom_digit_patches]
    return str(sum(top_digit_values) + sum(bottom_digit_values))
```

✗

MNLogic-3dgt

GPT-4o

```
def execute_command(image):
    image_patch = ImagePatch(image)

    # Step 1: Find the handwritten digits
    digit_patches = image_patch.find("digit")

    # Step 2: Sort them from left to right based on horizontal position
    digit_patches.sort(key=lambda x: x.horizontal_center)

    # Step 3: Read each digit as text (assume output is string like '0' or '1')
    digits = []
    for patch in digit_patches:
        digit_str = patch.simple_query("What digit is this?")
        if digit_str.strip() not in {"0", "1"}:
            raise ValueError(f"Invalid binary digit: {digit_str}")
        digits.append(int(digit_str.strip()))

    # Step 4: Compute XOR
    y = digits[0] ^ digits[1] ^ digits[2]

    return str(y)
```

✓

CodeLlama-13B

```
def execute_command(image):
    image_patch = ImagePatch(image)
    binary_digit_patches = image_patch.find("binary digit")
    binary_digit_patches.sort(key=lambda x: x.horizontal_center)
    binary_digit_patch_0 = binary_digit_patches[0]
    binary_digit_patch_1 = binary_digit_patches[1]
    binary_digit_patch_2 = binary_digit_patches[2]
    binary_digit_0 = binary_digit_patch_0.simple_query("What is the
value of the binary digit?")
    binary_digit_1 = binary_digit_patch_1.simple_query("What is the
value of the binary digit?")
    binary_digit_2 = binary_digit_patch_2.simple_query("What is the
value of the binary digit?")
    return str(int(binary_digit_0, 2) ^ int(binary_digit_1, 2) ^
int(binary_digit_2, 2))
```

✓

KandLogic-3obj

GPT-4o

```
def execute_command(image):
    image_patch = ImagePatch(image)
    object_patches = image_patch.find("object")

    # Dictionary to track color consistency for each shape
    shape_to_colors = {}

    for patch in object_patches:
        shape = patch.simple_query("What is the shape?")
        color = patch.simple_query("What is the color?")
        if shape not in shape_to_colors:
            shape_to_colors[shape] = set()
        shape_to_colors[shape].add(color)

    # Check if any shape has more than one color
    for colors in shape_to_colors.values():
        if len(colors) > 1:
            return "False"

    return "True"
```

✓

CodeLlama-13B

```
def execute_command(image):
    image_patch = ImagePatch(image)
    object_patches = image_patch.find("object")
    object_patches.sort(key=lambda x:
x.horizontal_center)
    object_patch_1 = object_patches[0]
    object_patch_2 = object_patches[1]
    object_patch_3 = object_patches[2]
    if object_patch_1.verify_property("object", "color")
== object_patch_2.verify_property("object", "color") ==
object_patch_3.verify_property("object", "color"):
        return "True"
    else:
        return "False"
```

✗

Figure 14: Programs generated by GPT-4o (**top row**) and CodeLlama-13B (**bottom row**) across different tasks. These programs are used by ViperGPT to decompose the visual deductive reasoning tasks. ✓ and ✗ indicate the correctness of the generated programs.