End-to-End Differentiable Optimization of Tool-Augmented Visual Reasoning via Probabilistic Graphs

Anonymous ACL submission

Abstract

Tool-augmented visual reasoning systems, exemplified by Visual Programming (VP), decompose complex visual questions into executable programs that invoke specialized visual models as tools, offering interpretable and compositional reasoning. However, these systems face a fundamental optimization challenge: the non-differentiable pipeline and the absence of intermediate supervision prevent end-to-end gradient-based optimization of invoked tools using only task-level labels. Existing methods either optimize the Large Language Model (LLM) planner while neglecting tool quality, or rely on teacher models and additional data collection, none achieving true end-to-end differentiable optimization. We propose **eDoT**, the first **e**nd-to-end **D**ifferentiable **o**ptimization method for **T**ool-augmented visual reasoning. Our key insight is that any program's variable dependency structure is isomorphic to a directed probabilistic graph. Applying Variable Elimination - an exact probabilistic inference technique - transforms the non-differentiable VP execution into a fully differentiable process, enabling gradient-based optimization with only task-level supervision, without additional data or teacher models. Experiments on GQA, NLVRv2 and Open Images demonstrate improvements of 3.7%, 7.6% and 20%, validating eDoT's effectiveness arcoss multiple visual models and cross-task generalization.

1 Introduction

LLM-based modular systems (Yao et al., 2023; Wang et al., 2024; Xi et al., 2025) have attracted increasing attention owing to their distinctive advantages: explicit reasoning structure, auditable execution processes, and the ability to tackle complex multi-step tasks without task-specific training. A prominent instantiation of this paradigm is **tool-augmented reasoning** (Schick et al., 2023; Hao et al., 2023; Qin et al., 2024), where an LLM planner orchestrates calls to specialized external models as tools (Shen et al., 2023), delegating domain-specific sub-tasks to capable experts and aggregating their outputs to produce a final answer. **Visual Programming** (**VP**) (Gupta and Kembhavi, 2023; Surís et al., 2023) exemplifies this approach in visual understanding: given a complex visual question, an LLM generates an executable program that invokes specialized visual tools - such as object detectors and visual question answering (VQA) models - to handle visual sub-tasks compositionally. We focus on this class of tool-augmented visual reasoning systems, where the final output is a discrete answer derived from the structured execution of visual sub-tasks, combining the interpretability of modular design and the flexibility of LLM-generated Programs.

However, despite these advantages, such systems face two compounding optimization challenges: the execution pipeline is non-differentiable preventing direct gradient-based optimization of invoked tool models; and intermediate sub-task supervision is unavailable, leaving only task-level labels to guide learning. The non-differentiability of VP execution stems from multiple sources: the discrete bounding box selection in localization modules, the deterministic image cropping conditioned on these discrete selections, and the symbolic logic operations in the EVAL aggregation module. While continuous relaxation techniques such as Gumbel-Softmax (Jang et al., 2017) can address discrete sampling in isolation, they cannot handle symbolic logic operations in EVAL, nor resolve the semantic ambiguity introduced by relaxing discrete spatial selections; moreover, they introduce approximation error through temperature annealing. Reinforcement learning (RL) (Sutton and Barto, 2018; Mnih et al., 2015) offers an alternative, but although it has been applied to optimize the LLM planner in VP (Khan et al., 2024), applying it to tool model optimization suffers from severe credit assignment difficulties in multi-step execution pipelines and

low sample efficiency due to sparse binary reward signals. These challenges have prompted two lines of work to enhance VP, yet as we discuss next, neither line achieves true end-to-end differentiable optimization.

Existing efforts to enhance VP fall into two categories: improving the LLM planner, and optimizing the invoked visual tool models - though the latter remains largely underexplored. For the first category, earlier works improve program generation through feedback mechanisms (Gao et al., 2024a), reinforcement learning (Khan et al., 2024), and dynamic RL-based planning control (Ke et al., 2024); more recently, DWIM (Ke et al., 2025) enables the LLM to identify tool failures and adjust its planning through tool-aware fine-tuning, while ViUniT (Panagopoulou et al., 2025) filters out erroneous programs by automatically generating visual unit tests. However, all these methods neglect the optimization of invoked visual tool models, which account for the majority of the errors in VP execution, as evidence by our experimental observations (Fig. 4). The only attempts to optimize visual tool models - SDVP (Wan et al., 2025), which relies on teacher-generated pseudo-labels, and CLOVA (Gao et al., 2024b), which requires additional internet data collection - both bypass true end-to-end gradient optimization. To date, no existing method achieves end-to-end differentiable optimization of the entire modular system using only task-level labels. This motivates a fundamentally different approach to tool-augmented visual reasoning: rather than circumventing the non-differentiability of the execution process, we seek to transform it into a differentiable one by exploiting the inherent structure of program execution.

In this paper, we propose **eDoT**, an **e**nd-to-end **D**ifferentiable **o**ptimization method for **T**ool-augmented visual reasoning. We observe that the execution of any VP program induces a natural dependency structure: each variable in the program is determined by specific input variables and the parameters of the invoked tool models, forming directed dependency relationships across the entire execution. Our key insight is that this dependency structure **is isomorphic to a directed probabilistic graph**, where nodes represent program variables and model parameters, and directed edges encode probabilistic dependencies. This isomorphism provides a principled foundation for applying probabilistic inference to the program execution process.

Concretely, we construct this directed probabilistic graph from the visual program and apply **Variable Elimination** (Shachter, 1986)—an exact probabilistic inference technique (Pearl, 1988)—to compute differentiable probability expressions for the program's output. This transforms the originally non-differentiable VP execution into an equivalent, fully differentiable probabilistic inference process on the graph, through which gradients flow precisely to the parameters of every invoked tool model. As a result, standard gradient-based optimization can be directly applied to fine-tune tool models using only task-level labels, without requiring teacher models (Wan et al., 2025), pseudo-labels, or additional data collection (Gao et al., 2024b). Crucially, because VP programs are of limited length and the resulting probabilistic graphs are of bounded scale, exact inference remains computationally tractable—yielding precise gradient signals rather than the approximations introduced by relaxation-based methods. To further stabilize optimization across programs of varying complexity, we propose a curriculum learning strategy that arranges training samples by program length, progressively introducing more complex multi-step programs as learning proceeds.

Note that end-to-end Vision-Language Models (VLMs) (e.g., GPT-4V (OpenAI, 2023), InternVL (Chen et al., 2024b)) and VP frameworks represent complementary paradigms with different design objectives—the former optimizes perceptual performance while the latter prioritizes interpretable, auditable reasoning. Our eDoT addresses the orthogonal question of how to optimize tool models within the VP paradigm, and we therefore compare only against VP enhancement methods of the same type (enhancing the capabilities of invoked visual tool models) under identical experimental conditions.

In summary, our contributions are as follows:

- **We propose eDoT, the first end-to-end differentiable optimization method for tool-augmented visual reasoning frameworks.** eDoT enables gradient-based fine-tuning of invoked tool models using only task-level labels, without requiring teacher models, pseudo-labels, or additional data collection.

- **We establish a principled isomorphism between VP execution and directed probabilistic graphical models.** This transforms the non-differentiable VP execution into an equivalent differentiable inference process,

through which precise gradient signals propagate to all invoked tool model parameters.

- **Extensive experiments validate the effectiveness and generalization of our eDoT.** On GQA (Hudson and Manning, 2019), NLVRv2 (Suhr et al., 2019), and Open Images (Shao et al., 2024), eDoT achieves improvements of 3.7%, 7.6%, and 20.0% with consistent gains across multiple visual models and cross-task generalization.

2 Related Works

Tool-Augmented Reasoning and Visual Programming. Tool-augmented reasoning (Schick et al., 2023; Hao et al., 2023; Qin et al., 2024)—where a central controller decomposes complex tasks and delegates sub-tasks to specialized external modules—has long been explored for interpretable and compositional AI. Early neuro-symbolic methods (Yi et al., 2018; Mao et al., 2019; Amizadeh et al., 2020) instantiated this paradigm through pre-designed Domain-Specific Languages (DSLs), but their fixed vocabularies limited generalization and flexibility. The emergence of LLMs (Zhao et al., 2023; Yang et al., 2024) transformed this landscape: Visual Programming frameworks (Gupta and Kembhavi, 2023; Surís et al., 2023) replaced hand-crafted DSLs with LLM-generated executable programs that dynamically invoke specialized visual models as tools, enabling zero-shot generalization to diverse visual reasoning tasks without task-specific training.

Methods for Enhancing Visual Programming. VP enhancement methods fall into two categories. For LLM planner improvement, prior works adopt execution feedback (Gao et al., 2024a), reinforcement learning (Khan et al., 2024), and DQN-based meta-control (Ke et al., 2024); more recently, DWIM (Ke et al., 2025) enables tool-failure-aware replanning via instruction-masking fine-tuning, while ViUniT (Panagopoulou et al., 2025) filters erroneous programs using automatically generated visual unit tests. For visual tool model improvement—the category most relevant to our work—SDVP (Wan et al., 2025) constructs a stepwise distillation pipeline in which a fully-supervised teacher provides pseudo-labels for each VP sub-task, while CLOVA (Gao et al., 2024b) localizes erroneous perception steps and collects targeted internet data to retrain the responsible module. Notably, all these methods treat the VP execution pipeline as a fixed, non-differentiable structure, and none attempts to propagate gradients through the full execution process using only task-level supervision—the gap that our eDoT addresses.

3 Methodology

In this section, we will introduce our eDoT in detail. Firstly, we will formulate the execution process of VP. Then, our eDoT will be introduced in three main parts: 1) the construction of the probabilistic graph corresponding to the visual program of VP; 2) the exact inference process on the constructed probabilistic graph, which can help transform the non-differentiable visual programming execution process into a differentiable one; 3) the loss function of utilizing original VR labels supervising VP outcomes.

3.1 Execution Process of VP

The core idea of VP in solving VR tasks is to use LLMs to generate a visual program to address the original VR question step-by-step. Then the visual program is executed, invoking different visual models as needed, to obtain the answer to the original VR question.

For each case in a VR task T, the input and label can be denoted as $< I, Q >$ and Y, respectively, where I is the image (For GQA and Open Images, one image, NLVRv2 two images), Q is the original VR question, and Y is the corresponding label. For convenience, we use VisProg (Gupta and Kembhavi, 2023), a classical VP framework, for further introduction. VisProg uses an LLM to accept the VR question Q as input and generates a visual program Pr as:

$$Pr = f(Pro, Q; \theta_{llm}), \tag{1}$$

where $f(\cdot; \theta_{llm})$ means an LLM with θ_{llm} as its weight parameters and Pro means the prompt, which includes the instruction and a few demonstrations. As shown in the top-left part of Fig. 1, the Visual Program Pr is an executable program with multiple lines. There are five kinds of modules in Pr: LOC (the NLVR task does not have the LOC module), CROP, VQA, EVAL, and RESULT. LOC and VQA are visual modules that invoke an object detection model and a VQA model, respectively. The number of the LOC or VQA modules in Pr could be more than one. The process of the i-th LOC module can be denoted as:

$$B^i = f(I, O^i; \theta_{loc}), \tag{2}$$

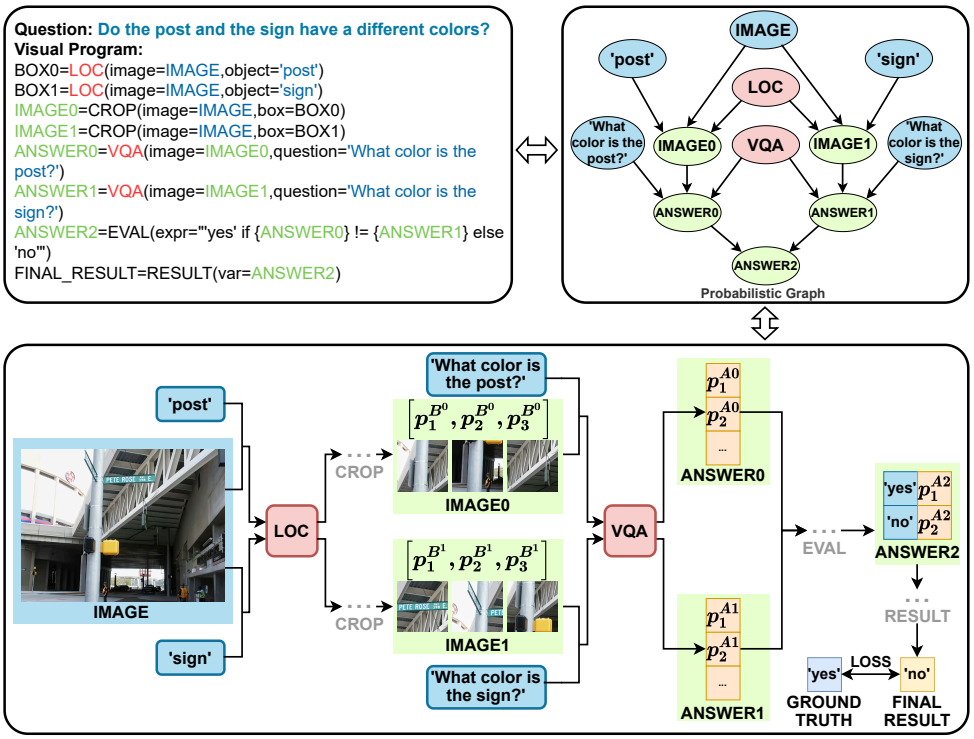

Figure 1: The overview of our proposed eDoT method. The top-left part is the visual program, and the top-right part is the constructed Probabilistic Graph. Below is the corresponding flowchart.

where O^i (in Fig. 1, 'post' or 'sign') denotes the object name needed to be detected, B^i denotes the bounding box of O^i in I, and $f(\cdot;\theta_{loc})$ denotes the invoked object detection model which with θ_{loc} as weight parameters. Then the sub-image I^i (in Fig. 1, 'IMAGE0' or 'IMAGE1') is produced through:

$$I^i = \text{CROP}(I, B^i). \tag{3}$$

I^i will then be input into the i-th VQA module to produce the sub-answer A^i of the sub-questions Q^i in this module as:

$$A^i = f(I^i, Q^i; \theta_{vqa}), \tag{4}$$

where $f(\cdot;\theta_{vqa})$ is the invoked VQA model with θ_{vqa} as its weight parameters. All N ($N \geq 1$) sub-answers $\{A^i\}_{i=0}^{N-1}$ will be sent into the EVAL module to produce the final answer A^f (in Fig. 1, 'ANSWER2') as:

$$A^f = \text{EVAL}(\{A^i\}_{i=0}^{N-1}), \tag{5}$$

where EVAL is a nested Boolean expression or a conditional selection statement that uses a nested Boolean expression as its condition. For the formal details of EVAL, please refer to Appendix A.1.

3.2 Details of Our eDoT

Our goal is to enhance VP for Task T by optimizing θ_{loc} and θ_{vqa} of visual tool models invoked by LOC and VQA modules with label Y of T as supervision. To achieve this goal, we transform the non-differentiable execution process of the visual program Pr into a differentiable probabilistic inference process on the probabilistic graph constructed from Pr.

Probabilistic Graph Construction.

As established in Sec. 1, the variable dependency structure of any VP program Pr is isomorphic to a directed probabilistic graph, which provides the foundation for applying exact probabilistic inference to achieve differentiable optimization. We construct this graph (Fig. 1, top-right) with three types of nodes: input variables (blue), intermediate or final output variables (green), and visual model weight parameters θ_{loc}, θ_{vqa} (red). Directed edges run from input variable nodes to their corresponding output variable nodes, and from weight parameter nodes to the corresponding model output nodes, reflecting that visual model outputs depend jointly on inputs and weights. Modules that do not alter probability distributions (CROP and RESULT) are omitted from the graph.

Exact Inference. Because the visual program Pr is generated from one VR question, the length of Pr is limited, and thus the scale of the probabilistic graph derived from Pr is limited. Considering those, we utilize "Variable Elimination" (Shachter,

1986), an Exact Inference technique (Pearl, 1988), to compute and get the differentiable probability expression of the outcome nodes. This probability serves as the answer prediction of A^f for the VR question Q. The expression includes the weight parameters of invoked visual models. As the case shown in Fig. 1, from the directed probabilistic graph, when the original inputs and weight parameters (all the blue and red nodes in the generated probabilistic graph) are given as condition, the variables ${A^i}_{i\geq 0}$ are "conditionally independent" (in Fig. 1 ANSWER0 and ANSWER1 are "conditionally independent"). So we can firstly calculate the probability distribution P^{A^i} of each A^i respectively, then utilize those distributions to calculate the distribution of the final answer prediction A^f (such as ANSWER2 in Fig. 1) according to the boolean condition formulation in EVAL. $p_j^{A^i}$ is the j-th component of P^{A^i}, represents the probability of A^i on the j-th output candidate. $p_j^{A^i}$ can be expressed as follows:

$$\begin{aligned} p_j^{A^i} =& P(A^i = A_j^i | I, Pr, \theta_{loc}, \theta_{vqa}) \\ =& P(A^i = A_j^i | I, \{Q^i\}_{i=0}^{N-1}, \{O^i\}_{i=0}^{N-1}, \theta_{loc}, \theta_{vqa}) \\ =& \sum_k P(A^i = A_j^i, I^i = I_k^i | \\ & I, \{Q^i\}_{i=0}^{N-1}, \{O^i\}_{i=0}^{N-1}, \theta_{loc}, \theta_{vqa}), \end{aligned} \tag{6}$$

where $\{Q^i\}_{i=0}^{N-1}$ and $\{O^i\}_{i=0}^{N-1}$ respectively represent all N number of sub-question Q^i and object O^i in Pr, I_k^i represents the sub-image crop from the k-th bounding box candidate produced by the LOC model with input (I, O^i). θ_{loc} and θ_{vqa} represent the weight parameters of the LOC and VQA models, respectively. Because variables A^i and I^i are not independent, we can further obtain:

$$\begin{aligned} p_j^{A^i} =& \sum_k P(A^i = A_j^i | I^i = I_k^i, I, \{Q^i\}_{i=0}^{N-1}, \\ & \{O^i\}_{i=0}^{N-1}, \theta_{loc}, \theta_{vqa}) \\ & \cdot P(I^i = I_k^i | I, \{Q^i\}_{i=0}^{N-1}, \{O^i\}_{i=0}^{N-1}, \theta_{loc}, \theta_{vqa}). \end{aligned} \tag{7}$$

By simplifying Equation 7 according to the conditional dependencies in the probabilistic graph, we obtain that

$$\begin{aligned} p_j^{A^i} =& \sum_k P(A^i = A_j^i | I^i = I_k^i, Q^i, \theta_{vqa}) \\ & \cdot P(I^i = I_k^i | I, O^i, \theta_{loc}), \end{aligned} \tag{8}$$

$$P(A^i = A_j^i | I^i = I_k^i, Q^i, \theta_{vqa}) = f_j(I_k^i, Q^i; \theta_{vqa}), \tag{9}$$

where $f_j(I_k^i, Q^i; \theta_{vqa})$ represents the probability on the j-th output candidate of the VQA model with Input (I_k^i, Q^i). Because the CROP function does not change the probability, we use $p_k^{B^i}$ representing the probability of the k-th bounding box candidate produced by the LOC model with weights θ_{loc} under input (I, O^i) (also represented as $f_k(I, O^i; \theta_{loc})$). So we get that:

$$\begin{aligned} p_k^{B^i} &= P(I^i = I_k^i | I, O^i, \theta_{loc}) \\ &= P(B^i = B_k^i | I, O^i, \theta_{loc}) \\ &= f_k(I, O^i; \theta_{loc}). \end{aligned} \tag{10}$$

Finally, we can express $p_j^{A^i}$ as:

$$p_j^{A^i} = \sum_k f_j(I_k^i, Q^i; \theta_{vqa}) \cdot f_k(I, O^i; \theta_{loc}). \tag{11}$$

After we have obtained $\{P^{A^i}\}_{i=0}^{N-1}$, the differentiable probability expressions of all sub-answers, we pass them into the EVAL boolean condition formulation to get the final differentiable answer prediction distribution:

$$y_l = f_{eval}(\{P^{A^i}\}_{i=0}^{N-1}; \text{EVAL}(\{A^i\}_{i=0}^{N-1})), \tag{12}$$

where y_l represents the probability of the l-th candidate of VP answer prediction for VR question Q, $f_{eval}(\cdot; \text{EVAL})$ denotes probability calculated based on the EVAL Boolean expression. Please refer to Appendix A.1 for the details of f_{eval}.

Loss Function. After we get the probability expression y_l of the final outcome, we use the cross-entropy function to construct the loss as follows:

$$L = \frac{1}{m} \sum_m \sum_l Y_l \log y_l, \tag{13}$$

where Y_l denotes the label on the l-th dimension of the original label Y, which means the ground truth on this answer candidate, m is the total number of training cases. Because y_l is differentiable, the loss L is differentiable. Hence, it can be trained through gradient-based optimization methods (such as SGD) to enhance the performance of VisProg on the objective VR task T.

4 Experiments

Experimental Setting. We evaluate our eDoT under VisProg (Gupta and Kembhavi, 2023) (a popular VP framework) on three widely employed VR tasks: **GQA** (Hudson and Manning, 2019) and **NLVRv2** (Suhr et al., 2019), along with the **Open Images** split of Visual CoT (Shao et al., 2024) tasks (denoted as Open Images hereafter), all three

Method			GQA test-dev balanced	NLVRv2 public test	Open Images test
VisProg	Zero-shot (baseline)	BLIP_{hg}	48.6	68.3	35.6
		BLIP_{lavis}	51.0	73.6	44.9
		InternVL1.5	56.3	78.2	40.6
	Supervised	BLIP_{hg} (SDVP)	50.4(+1.8)	75.2(+6.9)	-
		BLIP_{lavis} (Our **eDoT**)	54.7(+**3.7**)	**81.2**(+**7.6**)	64.9(+20.0)
		InternVL1.5 (Our **eDoT**)	**58.9**(+2.6)	80.5(+2.3)	**85.1**(+**44.5**)

Table 1: Results of different methods on GQA, NLVRv2, and Open Images. Our eDoT is effective for both BLIP (Li et al., 2022) and InternVL1.5-2B (Chen et al., 2024a). BLIP_{hg} refers to the Hugging Face (Jain, 2022) implementation of BLIP , and BLIP_{lavis} refers to the LAVIS (Li et al., 2023) implementation of BLIP. There are slight performance differences between the two libraries. SDVP (Wan et al., 2025) is another learning method for VisProg.

require real-world visual perception and compositional reasoning abilities. Each case in GQA and Open Images is constructed with one real-world image and a question asking about this image, while NLVRv2 is constructed with two real-world images and a statement to be determined as True or False. Following the prompt of VisProg (shown in Appendix A.5), we use DeepSeek-V3 (Liu et al., 2024) to generate visual programs for the cases in a subset of train balanced in GQA, and whole train splits in NLVRv2 and Open Images, which are then executed to produce the predicted answer. We collect the programs that are successfully compiled from the training set as our training data. Accuracy is reported respectively on the test-dev balanced of GQA, public test split of NLVRv2, and test split of Open Images. Following VisProg, we use Owl-ViT-large (Minderer et al., 2022) as the LOC model, and BLIP (Li et al., 2022) as the VQA model. Besides, we also evaluate our eDoT by adopting a more recent model, InternVL1.5-2B (Chen et al., 2024a), as the VQA model. For more Implementation Details, please refer to Appendix A.2.

4.1 Main Results

We compare the performance of our eDoT with an SOTA VP-enhanced method, SDVP (Wan et al., 2025), which focuses on improving VP-invoked visual models. Note that we have not compared our eDoT with CLOVA (Gao et al., 2024b) since the size and content of the training data are very different and hard to compare fairly. The major difference is that our eDoT optimizes a visual model invoked by an explicit multi-step process based on distant outcome supervision (original VR labels). In contrast, CLOVA optimizes a single-step process, allowing it to optimize vision models without

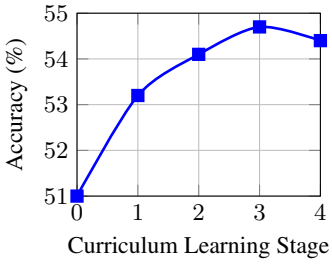

Figure 2: The accuracy of each curriculum learning stage while learning on GQA with our eDoT.

the need for distant supervision. CLOVA first attempts to pinpoint the erroneous step, then gathers additional labeled internet data specifically for optimizing that step, directly supervising the model at that particular stage. Additionally, CLOVA only reports results based on 500 randomly sampled cases, whereas we have tested our eDoT under a more general setup, evaluating its performance across the entire GQA test-dev balanced set, NLVRv2, and Open Images public test set. From the results reported by CLOVA on 500 random samples, the performance on NLVRv2 is significantly lower than Baseline in Tab. 1 (65.6 vs 73.6).

Comparison results on the three tasks are shown in Tab. 1. For the supervised methods under VisProg, we get answers by executing visual programs. eDoT with BLIP achieves improvements on GQA (+3.7%), NLVRv2 (+7.6%), and Open Images (+20.0%). SDVP achieves an increase of 1.8% and 6.9% on GQA and NLVRv2, respectively. Compared to SDVP, our method shows a greater improvement over the baseline, which validates the advantages of our method: achieving a larger performance gain without the need for an additional teacher model. Moreover, our eDoT remains effective when applied to the more powerful visual models. InternVL1.5-2B obtains a 2.6% improve-

Model	CL	GQA test-dev balanced	NLVRv2 public test
BLIP	Baseline	51.0	73.6
	✓	54.7(+3.7)	81.2(+7.6)
	✗	53.3(+2.3)	79.3(+5.7)

Table 2: Results comparison with different training paradigms. “CL” means “Curriculum Learning”.

Disrupted Programs	GQA test-dev balanced	NLVRv2 public test
Baseline	51.0	73.6
0%	54.7(+3.7)	81.8(+8.2)
20%	54.6(+3.6)	80.9(+7.3)
50%	53.6(+2.6)	77.8(+4.2)

Table 3: Effect of Disrupted Visual Programs Proportion on BLIP Model Performance. The batch size for NLVRv2 and GQA is **32**.

ment on GQA, a 2.3% improvement on NLVRv2, and a 44.5% improvement on Open Images, demonstrating the adaptability of our eDoT to the model.

4.2 Ablation Studies and Robustness Analysis

Training Paradigms. Initially, our experiments show that training the model with all data simultaneously barely enhances its performance, likely due to the need to handle complex task variations from the start, which can destabilize training before simple patterns are learned.

Thus, we adopt curriculum learning, starting with simpler tasks and progressively introducing more complex ones. We organize the training data by the number of visual program steps, beginning with one-step visual programs (only calculating steps invoking visual models), then moving to two-step programs, and gradually increasing the complexity of the training tasks. The experiment results in Tab. 2 indicate that using a curriculum learning approach can unleash the potential of our method. Without curriculum learning, less improvement in model performance is observed. With curriculum learning, our eDoT shows a 3.7% increase on GQA and a 7.6% increase on NLVRv2. In Fig. 2, we present the performance enhancements of VP across different stages of CL when trained on GQA under our eDoT. The effectiveness of VP reaches saturation by the third stage, suggesting that a total of 3 to 4 steps is an appropriate arrangement for the curriculum learning stage of our eDoT.

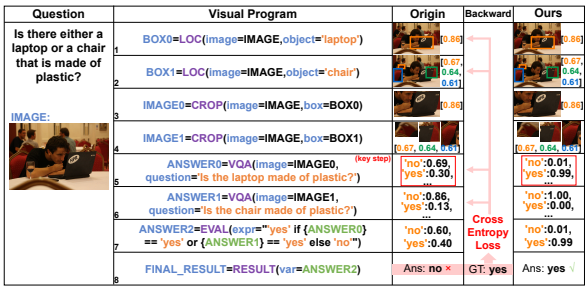

Figure 3: Qualitative example showing the program execution flow of VisProg original and after learning with our method. We provide detailed process outputs of each step during program execution. Ans means the final answer, and GT means the ground truth. For more cases, please refer to our supplementary materials.

Robustness to Noisy Visual Program. Our eDoT is an optimization method based on an explicit workflow, which is determined by visual programs generated by LLMs. Therefore, the quality of the visual program has a crucial impact on the performance of our method. However, the programs generated by LLMs can not be perfectly accurate, which necessitates that our method be robust to noisy data. To evaluate the robustness of our method against the noise, we introduce noise into the training dataset by intentionally disrupting the visual programs generated by LLMs. The types of disruptions include: 1) randomly replacing module input parameters and 2) replacing certain modules with alternatives. It is worth mentioning that these disruptions will not cause program execution workflow errors. We randomly select x% of the data for disruptions. As shown in Tab. 3, our method performs relatively well even when 20% of the training data is corrupted, with only a 0.1% drop in performance in GQA compared to that where no visual program is corrupted. Only when the artificial corruption rate rises to 50%, the performance of our eDoT drops from our best performance, with a 1.1% decrease in GQA and a 4.0% decrease in NLVRv2. These results demonstrate that our eDoT has robustness against the noise of the visual program. Moreover, combining Tab. 3 and the error analysis in (Gupta and Kembhavi, 2023) and Sec. 4.4, it is evident that current LLMs meet the quality requirement of generating visual programs.

4.3 Analysis of Generalization and Adaptability

Cross-task generalization ability. To evaluate the generalization ability of our eDoT, we conduct

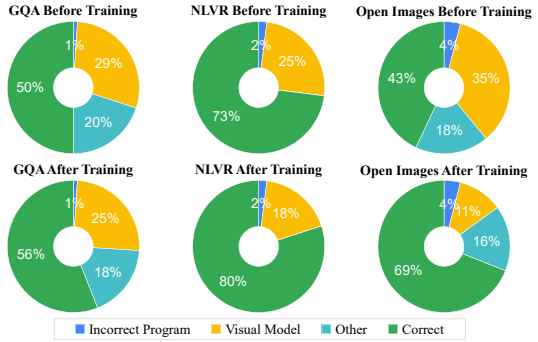

Figure 4: Sources of errors in VisProg before and after eDoT. We categorize the error causes into three groups: (1) Incorrect Program, (2) incorrect outputs from Visual Models, and (3) Other factors such as semantically similar answers that fail to match the ground truth.

Method			GQA (in-task)	Open Images (cross-task)
VLM	Zero-shot	InternVL1.5	63.2	41.0
	Supervised	InternVL1.5 (SFT)	64.6(+1.1)	42.7(+1.7)
VisProg	Zero-shot	InternVL1.5	56.3	40.6
	Supervised	InternVL1.5 (SDVP)	57.7(+1.4)	42.5(+1.9)
		InternVL1.5 (Our **eDoT**)	58.0(**+1.7**)	**44.7(+4.1)**

Table 4: Cross-task evaluation: InternVL1.5 (Chen et al., 2024a) trained on GQA under demonstrates improved accuracy when evaluated on Open Images, indicating enhanced cross-task generalization. Our eDoT demonstrates stronger cross-task generalization capabilities compared to SDVP (Wan et al., 2025) or SFT.

a cross-task evaluation by training the model on the GQA train split and evaluating it on the Open Images test set. As shown in Tab. 4, our eDoT achieves a 4.1% improvement in cross-task test accuracy, outperforming the 1.7% gain obtained with the End-to-end (without VisProg) Supervised Fine-Tuning (SFT) method. It can be observed that the performance improvement of SDVP (Wan et al., 2025) (another VP-enhanced method) on GQA after training (+1.4%) is lower than the improvement achieved by our eDoT. Additionally, SDVP's cross-task generalization ability on Open Images (+1.9%) is also lower than our eDoT's improvement. We attribute this difference to the fact that our eDoT makes better use of the dataset's result labels for learning, effectively avoiding the noise introduced by SDVP during the intermediate step supervision from the teacher model. These results demonstrate the solid cross-task generalization ability of our eDoT.

Robustness on longer programs and newer models. To further evaluate the robustness of our eDoT on longer programs and more recent models, we perform two additional experiments on GQA and NLVRv2. First, we train VP with BLIP as its VQA tool model exclusively on long programs from NLVRv2 (6–8 visual model steps, 8,000 samples, about 10% of the full training set), achieving a +4.7% accuracy gain (73.6%→78.3%). Second, we replace the visual models with more recent alternatives—LLMDet Swin-L (Fu et al., 2025) as the LOC tool model and InternVL3-2B (Zhu et al., 2025) as the VQA tool model—and apply eDoT on GQA, yielding a consistent +2.6% improvement (56.2%→58.8%), comparable to the gains observed with Owl-ViT-large (Minderer et al., 2022) and InternVL1.5-2B (Chen et al., 2024a). Notably, swapping in newer models alone does not improve the VP baseline, whereas eDoT reliably delivers gains regardless of the underlying model choice. These experimental results demonstrate that our eDoT is robust against longer programs and more recent models.

4.4 Qualitative Analysis

Error Analysis. We randomly sample 100 examples per dataset for manual inspection. As shown in Fig. 4, the dominant error source in VisProg is visual model failure, with very few incorrect programs—confirming the adequacy of LLM-generated programs and underscoring the importance of visual model optimization. eDoT reduces visual model error rates by 4%, 7%, and 24% on GQA, NLVRv2, and Open Images, respectively.

Case Study. Fig. 3 illustrates a representative GQA example where the baseline VQA model incorrectly ranks a key intermediate sub-answer (0.69 for "no" vs. 0.30 for "yes"), propagating an error to the final output. After eDoT optimization, the correct answer is ranked first (0.99 for "yes"), yielding the correct final result. More cases are provided in Appendix A.3.

5 Conclusion

We present eDoT, the first end-to-end differentiable optimization method for tool-augmented visual reasoning. By establishing an isomorphism between VP execution and directed probabilistic graphs, eDoT transforms the non-differentiable execution process into a differentiable one via exact inference, enabling gradient-based optimization using only task-level labels. Experiments on GQA, NLVRv2, and Open Images validate consistent improvements across multiple visual models and cross-task settings. Future research will extend eDoT beyond Visual Reasoning.

Limitations

The current framework is designed for visual reasoning tasks where outputs can be represented as probability distributions over discrete answer candidates. Extending eDoT to tasks with structured or continuous outputs, such as visual grounding or image generation, requires developing appropriate probabilistic formulations for non-discrete output spaces, which we identify as an important direction for future work.

References

Saeed Amizadeh, Hamid Palangi, Alex Polozov, Yichen Huang, and Kazuhito Koishida. 2020. Neuro-symbolic visual reasoning: Disentangling "visual" from "reasoning". In *International Conference on Machine Learning*, pages 279–290. Pmlr.

Zhe Chen, Weiyun Wang, Hao Tian, Shenglong Ye, Zhangwei Gao, Erfei Cui, Wenwen Tong, Kongzhi Hu, Jiapeng Luo, Zheng Ma, and 1 others. 2024a. How far are we to gpt-4v? closing the gap to commercial multimodal models with open-source suites. *Science China Information Sciences*, 67(12):220101.

Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, and 1 others. 2024b. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 24185–24198.

Shenghao Fu, Qize Yang, Qijie Mo, Junkai Yan, Xihan Wei, Jingke Meng, Xiaohua Xie, and Wei-Shi Zheng. 2025. Llmdet: Learning strong open-vocabulary object detectors under the supervision of large language models. *arXiv preprint arXiv:2501.18954*.

Minghe Gao, Juncheng Li, Hao Fei, Liang Pang, Wei Ji, Guoming Wang, Zheqi Lv, Wenqiao Zhang, Siliang Tang, and Yueting Zhuang. 2024a. De-fine: De composing and re fin ing visual programs with auto-feedback. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 7649–7657.

Zhi Gao, Yuntao Du, Xintong Zhang, Xiaojian Ma, Wenjuan Han, Song-Chun Zhu, and Qing Li. 2024b. Clova: A closed-loop visual assistant with tool usage and update. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13258–13268.

Tanmay Gupta and Aniruddha Kembhavi. 2023. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2023. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *Advances in neural information processing systems*, 36:45870–45894.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Drew A Hudson and Christopher D Manning. 2019. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6700–6709.

Shashank Mohan Jain. 2022. Hugging face. In *Introduction to transformers for NLP: With the hugging face library and models to solve problems*, pages 51–67. Springer.

Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*.

Fucai Ke, Zhixi Cai, Simindokht Jahangard, Weiqing Wang, Pari Delir Haghighi, and Hamid Rezatofighi. 2024. Hydra: A hyper agent for dynamic compositional visual reasoning. In *European Conference on Computer Vision*, pages 132–149. Springer.

Fucai Ke, Vijay Kumar B G, Xingjian Leng, Zhixi Cai, Zaid Khan, Weiqing Wang, Pari Delir Haghighi, Hamid Rezatofighi, and Manmohan Chandraker. 2025. Dwim: Towards tool-aware visual reasoning via discrepancy-aware workflow generation & instruct-masking tuning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3378–3389.

Zaid Khan, Vijay Kumar BG, Samuel Schulter, Yun Fu, and Manmohan Chandraker. 2024. Self-training large language models for improved visual program synthesis with visual reinforcement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14344–14353.

Dongxu Li, Junnan Li, Hung Le, Guangsen Wang, Silvio Savarese, and Steven C.H. Hoi. 2023. LAVIS: A one-stop library for language-vision intelligence. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 31–41, Toronto, Canada. Association for Computational Linguistics.

Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. 2022. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International conference on machine learning*, pages 12888–12900. PMLR.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B Tenenbaum, and Jiajun Wu. 2019. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *arXiv preprint arXiv:1904.12584*.

M Minderer, A Gritsenko, A Stone, M Neumann, D Weissenborn, A Dosovitskiy, A Mahendran, A Arnab, M Dehghani, Z Shen, and 1 others. 2022. Simple open-vocabulary object detection with vision transformers. arxiv 2022. *arXiv preprint arXiv:2205.06230*, 2.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, and 1 others. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Artemis Panagopoulou, Honglu Zhou, Silvio Savarese, Caiming Xiong, Chris Callison-Burch, Mark Yatskar, and Juan Carlos Niebles. 2025. Viunit: Visual unit tests for more robust visual programming. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 24646–24656.

Judea Pearl. 1988. Probabilistic reasoning in intelligent systems: Networks of plausible inference.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pages 9695–9717.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551.

Ross D Shachter. 1986. Evaluating influence diagrams. *Operations research*, 34(6):871–882.

Hao Shao, Shengju Qian, Han Xiao, Guanglu Song, Zhuofan Zong, Letian Wang, Yu Liu, and Hongsheng Li. 2024. Visual cot: Advancing multi-modal language models with a comprehensive dataset and benchmark for chain-of-thought reasoning. *Advances in Neural Information Processing Systems*, 37:8612–8642.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180.

Alane Suhr, Stephanie Zhou, Ally Zhang, Iris Zhang, Huajun Bai, and Yoav Artzi. 2019. A corpus for reasoning about natural language grounded in photographs. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6418–6428, Florence, Italy. Association for Computational Linguistics.

Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11888–11898.

Richard S Sutton and Andrew G Barto. 2018. Reinforcement learning: An introduction second edition. *Adaptive computation and machine learning: The MIT Press, Cambridge, MA and London*.

Wentao Wan, Nan Kang, Zeqing Wang, Zhuojie Yang, Liang Lin, and Keze Wang. 2025. A stepwise distillation learning strategy for non-differentiable visual programming frameworks on visual reasoning tasks. *Preprint*, arXiv:2309.09809.

Wentao Wan, Zeqing Wang, Nan Kang, Keze Wang, Zhiyu Shen, and Liang Lin. 2023. Visualprog distiller: Learning to fine-tune non-differentiable visual programming frameworks. *arXiv preprint arXiv:2309.09809*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, and 1 others. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, and 1 others. 2025. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101.

Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Shaochen Zhong, Bing Yin, and Xia Hu. 2024. Harnessing the power of llms in practice: A survey on chatgpt and beyond. *ACM Transactions on Knowledge Discovery from Data*, 18(6):1–32.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Josh Tenenbaum. 2018. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. *Advances in neural information processing systems*, 31.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, and 1 others. 2023.

A survey of large language models. *arXiv preprint arXiv:2303.18223*.

Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Hao Tian, Yuchen Duan, Weijie Su, Jie Shao, and 1 others. 2025. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *arXiv preprint arXiv:2504.10479*.

A Appendix

A.1 Formulation of "EVAL"

In VisProg (Gupta and Kembhavi, 2023), one kind of Visual Programming (VP) framework, after generating the answers to each sub-question, an EVAL module is used to combine those answers. We can express it as:

$$A^f = \text{EVAL}(\{A^i\}_{i=0}^{N-1}), \tag{14}$$

where A^f is the final answer produced by VisProg, $\{A^i\}_{i=0}^{N-1}$ represents the set of answers to sub-questions. EVAL is a nested Boolean expression or a conditional selection statement that uses a nested Boolean expression as its condition. The nested Boolean expression can be expressed as:

$$D^f = Bool(\{D^i\}_{i=0}^{N-1}), \tag{15}$$

where D^f is the final Boolean value from which to determine A^f, $\{D^i\}_{i=0}^{N-1}$ is a set containing multiple atomic Boolean expressions D^i, each D^i use A^i as a variable to calculate its Boolean value. When using our eDoT, the answer to each sub-question is represented as a probability distribution P^{A_i} on a candidate answer list. Then, we can get

$$\begin{aligned} y_l &= f_{eval}(\{P^{A^i}\}_{i=0}^{N-1}; \text{EVAL}(\{A^i\}_{i=0}^{N-1})) \\ &= f_{eval}(\{P^{A^i}\}_{i=0}^{N-1}; Bool(\{D^i\}_{i=0}^{N-1})). \end{aligned} \tag{16}$$

Here, we introduce the formulation of the EVAL function to combine those distributions into the probability distribution of the final answer. First, it is easy to calculate the Boolean probability distribution of each atomic Boolean expression D^i from P^{A^i} to get $P(D^i = T)$ and $P(D^i = F)$, in which 'T' represents 'True' and 'F' represents 'False'. There are 4 ways to combine the atomic Boolean expression D^i and D^s into the higher-level Boolean expression D:

$$D = \begin{cases} \neg D^i \\ D^i \text{ and } D^s \\ D^i \text{ or } D^s \\ D^i \text{ xor } D^s \end{cases}. \tag{17}$$

For $D = \neg D^i$,

$$\begin{aligned} P(D=T) &= P(\neg D^i = T) = P(D^i = F) \\ P(D=F) &= P(\neg D^i = F) = P(D^i = T). \end{aligned} \tag{18}$$

Visual Model	Curriculum Learning stage	Curriculum Learning program steps	Data Size GQA	Data Size NLVRv2
BLIP	1	≤ 1	15,000	9,681
	2	≤ 2	45,000	52,702
	3	≤ 3	60,000	54,866
	4	≤ 4	70,000	76,267

Table 5: The settings of curriculum learning. "Program steps" are defined as the total number of modules that invoke the visual model. The other modules (e.g., CROP, EVAL, and RESULT) that do not call the visual model are excluded from the count.

For $D = D^i$ and D^s,

$$\begin{aligned} P(D=T) &= P(D^i = T) \cdot P(D^s = T) \\ P(D=F) &= 1 - P(D=T). \end{aligned} \tag{19}$$

For $D = D^i$ or D^s,

$$\begin{aligned} P(D=T) &= 1 - P(D=F) \\ P(D=F) &= P(D^i = F) \cdot P(D^s = F). \end{aligned} \tag{20}$$

For $D = D^i$ xor D^s,

$$\begin{aligned} P(D=T) &= P(D^i = T) \cdot P(D^s = F) + P(D^i = F) \cdot P(D^s = T) \\ P(D=F) &= 1 - P(D=T). \end{aligned} \tag{21}$$

By repeatedly applying the above formula and calculating sequentially, we can ultimately obtain the probability distribution of the Boolean value for D^f, from which the probability distribution y of the final answer A^f for the original Visual Reasoning(VR) input $< I, Q >$ can be derived. Then we can use this answer prediction probability distribution y to construct the loss function with the label of this case (please refer to Sec. 3.2).

A.2 Implementation Details.

Following VisProg, we use Owl-ViT-large (Minderer et al., 2022) as the LOC model, and BLIP (Li et al., 2022) as the VQA model. Besides, we also evaluate our eDoT by adopting a more recent model, InternVL1.5-2B (Chen et al., 2024a), as the VQA model. For BLIP, while learning on GQA, we unfreeze the weights of the last 4 layers of its vision encoder and answer decoder; while learning on NLVRv2, we unfreeze the last 2 layers of its vision encoder and answer decoder because we think NLVRv2, with only True or False as the answer, is easier than GQA. For InternVL1.5-2B, we employ LoRA (Hu et al., 2022) to fine-tune its LLM component, which helps in controlling computational resources. We use AdamW (Loshchilov and Hutter, 2019) optimizer with learning rate 1×10^{-5}, batch

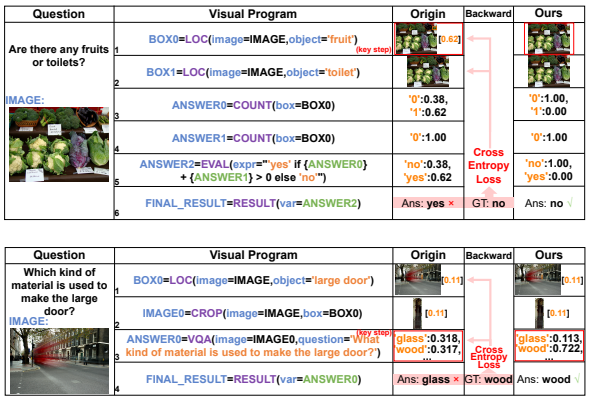

Figure 5: These program execution processes of the Visual Program before and after learning with our eDoT for a case in GQA. We provide detailed process outputs of each step during program execution. 'Ans' means the final answer, and 'GT' means the ground truth.

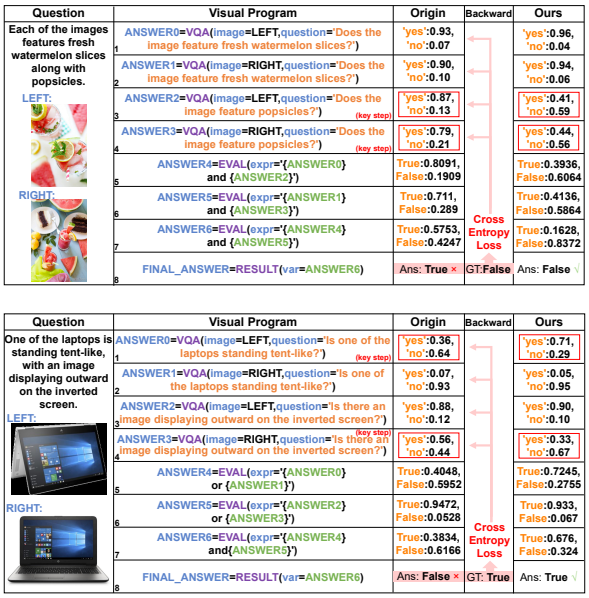

Figure 6: These program execution processes of the Visual Program before and after learning with our eDoT for a case in NLVRv2. We provide detailed process outputs of each step during program execution. 'Ans' means the final answer, and 'GT' means the ground truth.

size 32 on GQA and Open Images, 64 on NLVRv2 for training. In our experiments, we classify the difficulty of problems according to the length of program steps, and the samples with fewer steps are regarded as relatively easier ones. We apply a four-stage Curriculum Learning (CL) to our training process, which starts with samples of 1 visual module step and gradually progresses to more challenging ones of 4 visual module steps. The detailed curriculum learning settings are in Tab. 5.

A.3 More Cases

In Fig. 5, Fig. 6, and Fig. 7, we show more cases for our eDoT in the GQA (Hudson and Manning, 2019), NLVRv2 (Suhr et al., 2019), and Open Images (Shao et al., 2024) datasets. Fig. 5 upper subgraph shows a case in GQA. This case asks "Are there any fruits or toilets?", and there are no fruits or toilets in the image. So the ground truth is "no". However, the LOC model in the first step makes a mistake in locating a bounding box of "fruit". This mistake makes the final answer wrong. After optimizing the LOC model using our eDoT, the mistake of the LOC model in the first step has been corrected. The output of the LOC model in this step now produces no bounding box for the object "fruit", the final answer can also be corrected from "yes" to "no". Fig. 5 lower subgraph shows a case in GQA, which is a shorter, similar case to the case shown in Fig. 3.

Fig. 6 upper subgraph shows a case in NLVRv2. The statement is "Each of the images features fresh watermelon slices along with popsicles". The pretrained VQA model makes mistakes when answering "Does the image feature popsicles?" for both the left and right images. Hence, the final answer is wrong, because there are no popsicles in both images. After optimizing the VQA model using our eDoT, the outputs of the 1st and 2nd steps are still right, and the outputs of the 3rd and 4th steps are corrected: the probability of "no" is larger than "yes". Then the final answer has been corrected from the original wrong answer "True" to the right answer "False". Fig. 6 lower subgraph shows another case in NLVRv2.

Fig. 7 upper subgraph shows a case that asks "What is the people who holds the microphone doing?", the ground truth is "sitting". However, the VQA model in the fourth step makes a mistake in determining what people are doing. This mistake makes the final answer wrong. After optimizing the VQA model using our eDoT, the mistake in the fourth step of the VQA model has been corrected. The output of the VQA model in this step now produces the correct answer "sitting", the final answer can also be corrected from "talking" to "sitting". Fig. 7 lower subgraph shows another case in Open Images.

A.4 Why Learning under VisProg?

Our eDoT is fundamentally a technique for optimizing neural network models invoked under an

Figure 7: These program execution processes of the Visual Program before and after learning with our eDoT for a case in Open Images. We provide detailed process outputs of each step during program execution. 'Ans' means the final answer, and 'GT' means the ground truth.

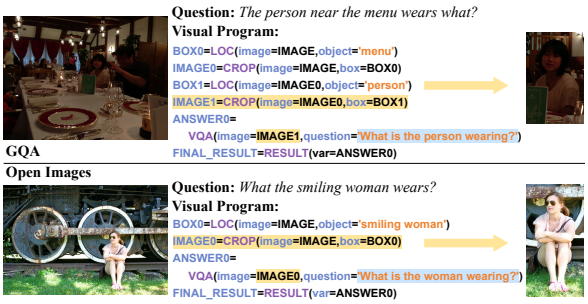

Figure 8: Example of sub-task module generalization. The top example is from the in-task GQA dataset, and the bottom is from the cross-task Open Images dataset. The VQA module learns to perform the sub-task "What is the person wearing" during GQA training, which generalizes naturally to similar queries such as "What is the woman wearing" in Open Images. Although the original images and questions differ significantly, the cropped regions after LOC and CROP operations (shown on the right) are more visually similar, which we think will help reduce input distribution shift and enhance cross-task generalization.

explicit procedure (here, the VisProg's visual program). Its key contribution lies in enabling end-to-end learning with only outcome supervision for non-differentiable yet interpretable VP frameworks. Compared to conventional end-to-end fine-tuning of pure neural networks, our eDoT is combined with explicit symbolic computation or reasoning, which preserves the interpretability advantage of visual programming.

Moreover, for compositional tasks where interfaces between the task and invoked visual modules mismatch (e.g., NLVRv2 takes two images as input, while its invoked VQA module processes only one), eDoT still permits optimization. This expands the scope of optimizable data to broader compositional task scenarios.

We further argue that the strategy of our eDoT, which learns under explicit visual programming, offers another advantage over pure neural network paradigms: stronger cross-task generalization. When decomposing complex vision tasks that involve intricate images and comprehension demands via visual programs, the resulting visual subtasks for model training become simpler, which contain localized images and partial understanding requirements. This narrows down the input distribution gaps of subtasks across different tasks, allowing visual models trained on one task to generalize better to others through explicit programmatic composition.

To evaluate the advantage of our eDoT for cross-task generalization, we train visual models on the GQA (Hudson and Manning, 2019) train set using both our eDoT within the VisProg (Wan et al., 2023) framework and a traditional end-to-end supervised fine-tuning approach based on pure neural networks (marked with "SFT"). We then evaluate their cross-task performance on the Open Images (Shao et al., 2024) test split. As shown in Tab. 4, our eDoT achieves an accuracy of 44.7% on Open Images, representing a 4.1% improvement over the baseline and outperforming the pure-network-training method, which achieves 42.7% on Open Images, only a 1.7% improvement. Additionally, we examine a hybrid approach "H_SFT" with the VisProg, invoking the visual model fine-tuned in a pure-network end-to-end supervised learning way. The "H_SFT" method achieves an accuracy of 43.5% under the same testing conditions on Open Images, an improvement of 2.9%, better than the "SFT" setting, yet still fell behind the performance of our eDoT. These results demonstrate that: i) evaluating with VisProg, which decomposes the original task, can improve the cross-task generalization performance; ii) training under VisProg can further improve that performance, which supports the cross-task generalization advantage of our method.

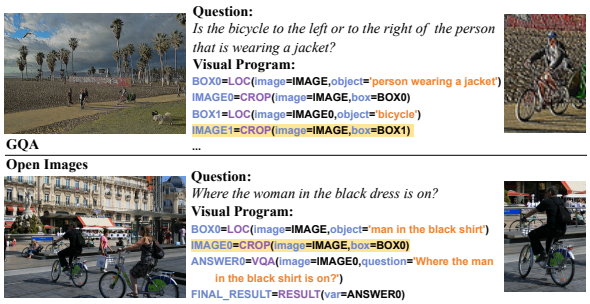

Figure 9: Example of visual input gap comparing. The top example is from the in-task GQA dataset, and the bottom is from the cross-task Open Images dataset. Although the original images and questions differ significantly, the cropped regions after LOC and CROP operations (shown on the right) are more visually similar, which we think will help reduce input distribution shift and enhance cross-task generalization.

We further provide a visual explanation about why learning under VisProg will benefit cross-task generalization through specific cases. For example, in one GQA training case illustrated in Fig. 8, the VQA sub-module within the visual program learns to answer questions such as "What is the person wearing?" This capability is easier to transfer to similar questions like "What is the woman wearing" in the Open Images than if learning with the original question as input (from "The person near the menu wears what" to "What the smiling woman wears"). Furthermore, the LOC and CROP operations applied to images help narrow down the input distributions between GQA and Open Images by focusing on consistent image regions of interest. As shown on the right side of Fig. 8 and Fig. 9, although the original images and questions in GQA and Open Images differ substantially, the localized and cropped subimages exhibit greater visual similarity, which aids in improving the cross-task generalization.

In contrast, the traditional end-to-end pure-network learning paradigm lacks explicit task structure modeling, which forces neural network models to learn mapping relationships directly from raw input-output distributions. Due to the compositional nature of the real-world data, the distribution of the original task usually exhibits greater divergence compared to that of sub-tasks in VisProg. Consequently, pure end-to-end learning with neural networks demonstrates relatively weaker cross-task generalization capability.

A.5 Prompts for Visual Programs Generation

We show the prompts used to generate the visual programs for GQA (Hudson and Manning, 2019), NLVRv2 (Suhr et al., 2019), and Open Images (Shao et al., 2024). Due to space limitations, only partial demonstrations are shown here for each Prompt. For the remaining demonstrations, please refer to the code.

Give the visual program to solve the problem. Only output the program. Do not output "'.
You can only use these modules in the program:
BOX=LOC(image=IMAGE,object="object")
ANSWER=COUNT(box=BOX)
IMAGE=CROP(image=IMAGE,box=BOX)
IMAGE=CROP_RIGHTOF(image=IMAGE,box=BOX)
IMAGE=CROP_LEFTOF(image=IMAGE,box=BOX)
IMAGE=CROP_INFRONTOF(image=IMAGE,box=BOX)
IMAGE=CROP_BEHIND(image=IMAGE,box=BOX)
IMAGE=CROP_BELOW(image=IMAGE,box=BOX)
IMAGE=CROP_ABOVE(image=IMAGE,box=BOX)
ANSWER=VQA(image=IMAGE,question="question")
ANSWER=EVAL(expr="expr")
FINAL_RESULT=RESULT(var=ANSWER)
Do not use any other modules!

Question: Do the post and the sign have a different colors?
Program:
BOX0=LOC(image=IMAGE,object="post")
IMAGE0=CROP(image=IMAGE,box=BOX0)
BOX1=LOC(image=IMAGE,object="sign")
IMAGE1=CROP(image=IMAGE,box=BOX1)
ANSWER0=VQA(image=IMAGE0,question="What color is the post?")
ANSWER1=VQA(image=IMAGE1,question="What color is the sign?")
ANSWER2=EVAL(expr="'yes' if {ANSWER0} != {ANSWER1} else 'no'")
FINAL_RESULT=RESULT(var=ANSWER2)

Question: What color is the curtain that is to the right of the mirror?
Program:
BOX0=LOC(image=IMAGE,object="mirror")
IMAGE0=CROP_RIGHTOF(image=IMAGE,box=BOX0)
ANSWER0=VQA(image=IMAGE0,question="What color is the curtain?")
FINAL_RESULT=RESULT(var=ANSWER0)

Question: Does the traffic cone have white color?
Program:
BOX0=LOC(image=IMAGE,object="traffic cone")
IMAGE0=CROP(image=IMAGE,box=BOX0)
ANSWER0=VQA(image=IMAGE0,question="Does the traffic cone have white color?")
FINAL_RESULT=RESULT(var=ANSWER0)

Question: **{new_question}**
Program:

Figure 10: Prompt for DeepSeek-V3 to generate Visual Programs of GQA.

Give the visual program to determine whether the statement is True or False. Only output the program. Do not output "'. Unless otherwise specified, the statement describes the combined situation in both images.
You can only use these modules in the program:
ANSWER=VQA(image=IMAGE,question="question")
ANSWER=EVAL(expr="expr")
FINAL_RESULT=RESULT(var=ANSWER)
Do not use any other modules!

Statement: there are at least seven wine bottles in the image on the left
Program:
ANSWER0=VQA(image=LEFT,question="How many wine bottles are in the image?")
ANSWER1=EVAL(expr='{ANSWER0} >= 7')
FINAL_ANSWER=RESULT(var=ANSWER1)

Statement: One dog is laying down.
Program:
ANSWER0=VQA(image=LEFT,question="How many dogs are laying down?")
ANSWER1=VQA(image=RIGHT,question="How many dogs are laying down?")
ANSWER2=EVAL(expr="{ANSWER0} + {ANSWER1} == 1")
FINAL_ANSWER=RESULT(var=ANSWER2)

Statement: An image shows exactly two seals in direct contact, posed face to face.
Program:
ANSWER0=VQA(image=LEFT,question="How many seals are in the image?")
ANSWER1=VQA(image=RIGHT,question="How many seals are in the image?")
ANSWER2=VQA(image=LEFT,question="Are the seals in direct contact?")
ANSWER3=VQA(image=RIGHT,question="Are the seals in direct contact?")
ANSWER4=VQA(image=LEFT,question="Are the seals posed face to face?")
ANSWER5=VQA(image=RIGHT,question="Are the seals posed face to face?")
ANSWER6=EVAL(expr="{ANSWER0} == 2 and {ANSWER2} and {ANSWER4}")
ANSWER7=EVAL(expr="{ANSWER1} == 2 and {ANSWER3} and {ANSWER5}")
ANSWER8=EVAL(expr="{ANSWER6} xor {ANSWER7}")
FINAL_ANSWER=RESULT(var=ANSWER8)

Statement: **{new_statement}**
Program:

Figure 11: Prompt for DeepSeek-V3 to generate Visual Programs of NLVRv2.

Give the visual program to solve the problem. Only output the program. Do not output “‘.
You can only use these modules in the program:
BOX=LOC(image=IMAGE,object=“object”)
ANSWER=COUNT(box=BOX)
IMAGE=CROP(image=IMAGE,box=BOX)
IMAGE=CROP_RIGHTOF(image=IMAGE,box=BOX)
IMAGE=CROP_LEFTOF(image=IMAGE,box=BOX)
IMAGE=CROP_INFRONTOF(image=IMAGE,box=BOX)
IMAGE=CROP_BEHIND(image=IMAGE,box=BOX)
IMAGE=CROP_BELOW(image=IMAGE,box=BOX)
IMAGE=CROP_ABOVE(image=IMAGE,box=BOX)
ANSWER=VQA(image=IMAGE,question=“question”)
ANSWER=EVAL(expr=“expr”)
FINAL_RESULT=RESULT(var=ANSWER)
Do not use any other modules!

Question: What is the people that holds the guitar doing?
Program:
BOX0=LOC(image=IMAGE,object=“people”)
IMAGE0=CROP(image=IMAGE,box=BOX0)
ANSWER0=VQA(image=IMAGE0,question=“Is this person holding a guitar?”)
ANSWER1=VQA(image=IMAGE0,question=“What is the people doing?”)
ANSWER2=VQA(image=IMAGE,question=“What is the people that holds the guitar doing?”)
ANSWER3=EVAL(expr=“{ANSWER1} if {ANSWER0} == ‘yes’ else {ANSWER2}”)
FINAL_RESULT=RESULT(var=ANSWER3)

Question: Who is on the plastic chair?
Program:
ANSWER0=VQA(image=IMAGE,question=“Who is on the plastic chair?”)
FINAL_RESULT=RESULT(var=ANSWER0)

Question: Where the standing boy is under?
Program:
BOX0=LOC(image=IMAGE,object=“standing boy”)
IMAGE0=CROP_ABOVE(image=IMAGE,box=BOX0)
ANSWER0=VQA(image=IMAGE0,question=“Where the standing boy is under?”)
FINAL_RESULT=RESULT(var=ANSWER0)

Question: **{new_question}**
Program:

Figure 12: Prompt for DeepSeek-V3 to generate Visual Programs of Open Images.