ADALOOP: THINKING WITH THE VISION ENCODER

Anonymous authors
Paper under double-blind review

ABSTRACT

Most vision-language models (VLMs) treat the vision encoder as a static feature extractor: it encodes an image once, independently of the question, and leaves subsequent reasoning to the language model. Yet different questions may require different visual operations and processing depths. We present AdaLoop, a framework for question-conditioned visual recurrence in VLMs. AdaLoop adapts visual computation to the question and reuses the encoder's transformations across multiple iterations, with the pretrained vision weights frozen. We also introduce VisRO, a procedural dataset and generation pipeline for training and evaluating vision-centric reasoning. On six VisRO tasks, AdaLoop outperforms standard fine-tuning and question-conditioned single-pass baselines with matched training data and optimizer updates. These gains hold across three VLM families on both in-distribution and out-of-distribution examples. Our findings suggest that pretrained vision encoders can contribute to reasoning through task-directed, iterative visual computation.

1 INTRODUCTION

Most vision–language models (VLMs) encode an image once, independently of the question, and pass the resulting visual tokens to a language model for subsequent reasoning. The same visual computation serves every question, whether the answer depends on a single local feature or on relating distant parts of the scene. A fixed pass may not perform the computation a task requires: useful visual features may lie at intermediate encoder layers (Bolya et al., 2025) or be lost along the way (Tong et al., 2024b; Ganz et al., 2024). The encoder thus acts only as a feature extractor, even though vision architectures can themselves learn to reason when trained from scratch (Hu et al., 2026; Shu et al., 2026). Visual reasoning, then, need not be left entirely to the language model.

For a pretrained VLM, an appealing possibility is to reuse the visual transformations already learned by its vision encoder. We draw inspiration from human vision, where processing is directed by the task (Ullman, 1984) and recurrent (Lamme & Roelfsema, 2000; Rao & Ballard, 1999). We investigate how to equip a pretrained vision encoder with both capabilities through (i) *question conditioning*, which modulates the encoder's features according to the task (Perez et al., 2018), and (ii) *looping (or recurrence)*, which reapplies weight-tied transformations to an evolving state (Dehghani et al., 2019; Geiping et al., 2025; Zhu et al., 2025).

Introducing these mechanisms into a pretrained VLM raises two questions. First, each encoder layer was trained to operate at a particular feedforward depth, so simply looping a span of its frozen layers yields little benefit (Section 5.2). Second, looping produces a sequence of visual states rather than a single output, and it is not obvious how to present the sequence to the language model.

We approach this as a design problem and present AdaLoop, a framework for adaptive visual recurrence in pretrained VLMs (Figure 1). AdaLoop creates a *perception loop* by repeatedly applying a span of the vision encoder to an evolving visual state, while keeping the original vision weights frozen. It uses question-conditioned adaptive layer normalization (AdaLN) (Perez et al., 2018; Peebles & Xie, 2023) to modulate the shared transformations at each iteration. We compare three ways for the language model to use the resulting sequence of states: reading only the current state, accumulating states across iterations, and interleaving states with answer drafts. All variants train only on triples of (image, question, answer), using the same final-answer target across visual iterations to support adaptive inference-time compute.

Evaluating vision-centric reasoning requires diagnostic tasks with systematic difficulty control and scalable training data. We introduce VisRO (Visual Routines) to support this research effort. VisRO is a procedural dataset and generation pipeline encompassing tasks like grouping, counting, connec-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

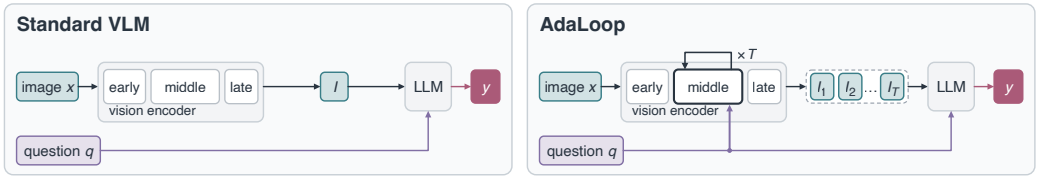


Figure 1: **Looking once versus repeatedly.** *Left:* A standard VLM encodes the image once, independently of the question. *Right:* AdaLoop turns a span of the vision encoder into a question-conditioned *perception loop*, updating its visual state over T iterations. The language model answers from the resulting sequence, using either the current state or retained history.

tivity, spatial tracing, and amodal perception—all of which demand careful visual inspection with minimal reliance on language priors or external knowledge (Figure 3). Our pipeline provides programmatically verifiable answers, controllable difficulty, and controlled distributional shifts in task structure or visual appearance for out-of-distribution evaluation. VisRO serves as reusable infrastructure for the community, and our AdaLoop experiments use six tasks from the collection.

Experiments with three VLM families, Qwen2.5-VL (Bai et al., 2025), PaliGemma 2 (Steiner et al., 2024) and LLaVA-1.5 (Liu et al., 2024a) show that with matched training data and optimizer updates, AdaLoop's components provide complementary gains. Question-conditioning establishes strong single-pass controls, and looping adds a further 2.8–4.6 percentage points of in-distribution VisRO accuracy, for a total improvement of about 11–15 points over standard fine-tuning. On Qwen2.5-VL, at matched parameter count and approximately matched training FLOPs, AdaLoop achieves competitive VisRO performance with better general visual question answering (VQA) capabilities than longer-trained single-pass controls. Our results suggest that existing visual transformations can be steered by the question and reused for iterative reasoning, even with the original weights frozen.

2 Related Work

Task-directed vision. Task-directed processing has roots in *visual routines*, combinations of basic operations selected by the task (Ullman, 1984). Language can supply this task signal through feature-wise modulation (Perez et al., 2018) or instruction-aware feature extraction (Dai et al., 2023), and prior work conditions vision encoders on language through normalization (de Vries et al., 2017; Liu et al., 2026a), cross-attention (Abramovich et al., 2024), or token injection (Ganz et al., 2024; Kim et al., 2026; Mao et al., 2026). AdaLoop applies such conditioning to encoder layers that are reused across iterations.

Recurrent and looped architectures. Recurrence has a long history in computational models (Pinheiro & Collobert, 2014; Liang & Hu, 2015; Liao & Poggio, 2016; Linsley et al., 2018). Looped Transformers increase effective depth by sharing parameters across iterations (Dehghani et al., 2019; Giannou et al., 2023; Lee & Park, 2026). Most are trained from scratch, either as task-specific reasoners for problems such as Sudoku, mazes, and ARC (Wang et al., 2025; Jolicoeur-Martineau, 2025; Baek et al., 2026; Shu et al., 2026) or as language models (Geiping et al., 2025; Zhu et al., 2025). Related work studies their scaling (Alabdulmohsin & Zhai, 2025; Lee et al., 2026; Wang et al., 2026c; Prairie et al., 2026; Schwethelm et al., 2026; Chen et al., 2026b), architectural design choices (Huang et al., 2026), adaptive stopping (Graves, 2016; Zhou et al., 2020; Bae et al., 2025b), and iteration-dependent modulation (Jeddi et al., 2026). Other work retrofits recurrence into pretrained language models (Bae et al., 2025a; McLeish et al., 2026; Park et al., 2026b), and recent multimodal systems loop all or part of their language backbones (Xu et al., 2026b; Zhang et al., 2026a; Park et al., 2026a). AdaLoop instead loops layers of the vision encoder in a pretrained VLM, and studies how the language model should use the resulting visual states (Appendix D).

Visual reasoning in VLMs. Visual reasoning methods differ in where intermediate visual computation takes place and how it is controlled. External image operations and visual prompting provide selected or annotated visual evidence for language-model reasoning (Shao et al., 2024a; Yang et al., 2023; Hu et al., 2024; Wu & Xie, 2024; Shen et al., 2025; Zheng et al., 2026). Other approaches perform intermediate computation within the language model using generated visual latents (Li et al., 2026a; Viveiros et al., 2026; Wang et al., 2026b) or learnable visual tokens (Li et al., 2026b). Language model states or generated sub-questions can also guide further visual processing (Yu et al., 2025; Galatolo et al., 2026; Zhang et al., 2026b). AdaLoop carries an internal vision-encoder state across iterations, updating it under the guidance of the original question.

Visual reasoning benchmarks. Existing benchmarks assess broad multimodal capabilities (Liu et al., 2024c; Chen et al., 2024), compositional reasoning (Johnson et al., 2017; Hudson & Manning, 2019), and reasoning about text in images (Singh et al., 2019). BLINK and CV-Bench probe core perceptual abilities (Fu et al., 2024; Tong et al., 2024a), while BabyVision emphasizes visual reasoning with limited reliance on language knowledge (Chen et al., 2026a). Scalable training resources for diverse vision-centric reasoning tasks remain limited. VisRO complements existing benchmarks with a procedural generation pipeline supporting both training and evaluation, with explicit controls over difficulty and distribution shifts.

3 METHOD

AdaLoop introduces question-conditioned recurrence into a pretrained vision–language model. We denote its vision encoder, connector, and language model by E, M, and G, respectively. Given an image x and a question q, AdaLoop repeatedly applies a group of consecutive layers in E to update a visual state, with each update conditioned on q. The later encoder layers and the connector M convert each updated state into visual tokens for G. We study two design choices: how to condition the visual updates on the question, and how the language model should use the visual tokens produced across iterations.

3.1 LOOPED VISION ENCODER

We organize the visual computation into three parts: a prelude $\mathcal{P}$, a loop body $\mathcal{R}$, and a coda $\mathcal{C}$. The prelude processes the image once to produce an initial visual state z_0. The loop body consists of consecutive encoder layers, augmented with question conditioning and reused across iterations. Let z_t denote the visual state after iteration t. The coda comprises the remaining encoder layers and the vision–language connector, mapping z_t to a sequence of visual tokens I_t for the language model,

$$z_0 = \mathcal{P}(x), \qquad z_t = \mathcal{R}(z_{t-1};\, q), \qquad I_t = \mathcal{C}(z_t), \qquad t = 1, \ldots, T. \tag{1}$$

The loop body shares its parameters across iterations. The state z_t carries information to the next visual iteration, while I_t provides the corresponding visual input to the language model.

We condition the loop body $\mathcal{R}$ on the question q through adaptive layer normalization (AdaLN) (Perez et al., 2018; Peebles & Xie, 2023). A learned attention pooler summarizes the question's input embeddings into a vector c_q. For each block ℓ, a module $A_\ell(c_q)$ predicts feature-wise shifts β_ℓ^f, scales γ_ℓ^f, and residual gains α_ℓ^f for each sublayer $f \in \{\mathrm{Attn}, \mathrm{MLP}\}$:

$$h \leftarrow h + (\mathbf{1} + \alpha_\ell^f) \odot f\Big((\mathbf{1} + \gamma_\ell^f) \odot \mathrm{LN}(h) + \beta_\ell^f\Big), \tag{2}$$

Here h denotes the visual state, LN layer normalization, and $\odot$ element-wise multiplication. Modulation is shared across tokens and iterations. Initializing $\alpha = \beta = \gamma = 0$ preserves the original encoder computation at the first training iteration for loop at $t = 1$.

Adaptation and recurrence are complementary. As we analyze in Appendix A, AdaLN and looping affect visual computation at different granularities. Intuitively, AdaLN provides question-dependent control within each block and feature channel, modifying both the inputs and residual contributions of visual transformations. Also, AdaLN's scaling of a residual update can locally approximate the first-order effect of repeating the transformation (Appendix A.2), suggesting layer-wise control over effective depth. Meanwhile, looping explicitly repeats a span of blocks, recomputing their updates as the visual state evolves. Together, they combine fine-grained adaptation with successive applications of the shared computation, and our experiments show benefits from both.

3.2 CONNECTING VISUAL ITERATION TO LANGUAGE

We consider three ways for the language model to use the visual tokens produced across iterations (Figure 2). These interfaces differ in whether earlier visual states and language predictions are retained in the context. For variant $v \in \{A, B, C\}$, let $\Pi_t^{(v)}$ denote the language-model prefix at iteration t. Here, q denotes the question embeddings, and $[\,\cdot \mid \cdot\,]$ denotes sequence concatenation.

Variant A: current visual state. A straightforward strategy is to provide only the visual tokens from the current iteration:

$$\Pi_t^{(A)} = [I_t \mid q].$$

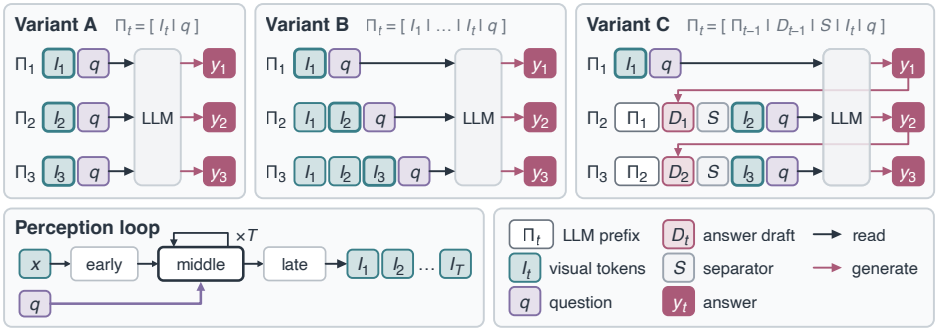


Figure 2: **Connecting visual iteration to language.** The perception loop produces visual tokens I_t across iterations, which are passed to the LLM by concatenation $[\,\cdot \mid \cdot\,]$ with question q. **A** generates intermediate and final answers y_t using each I_t in isolation, and **B** does so using the entire visual history $I_1 \dots I_t$. **C** interleaves each I_t with intermediate answers using separator S to distinguish rounds. In all cases, the stability of intermediate answers y_t can be used for adaptive halting.

This keeps the visual context fixed in length as loop depth increases. Information from earlier visual iterations reaches the language model through the recurrent state z_t.

Variant B: accumulated visual states. We can give the language model direct access to earlier visual states by concatenating their visual tokens:

$$\Pi_t^{(B)} = [I_1 \mid \cdots \mid I_t \mid q].$$

The language model can then use the full visual trajectory to predict the answer, therefore reducing the reliance on the current state. This relaxes the need to preserve earlier information in the recurrent state, at the cost of a visual context that grows linearly with loop depth.

Variant C: interleaved generation. Beyond retaining visual states, we can also retain the language model's intermediate predictions. Starting from $\Pi_1^{(C)} = [I_1 \mid q]$, the language model greedily generates a length-bounded answer draft D_{t-1} at each $t < T$. We append this draft and the next visual input to the prefix:

$$\Pi_t^{(C)} = [\Pi_{t-1}^{(C)} \mid D_{t-1} \mid S \mid I_t \mid q],$$

where S is a fixed separator. Subsequent predictions can thus use both the visual history and earlier answers, with additional context and generation cost. The drafts enter only the language context; visual updates continue to use the original question condition.

3.3 Deep Supervision for Adaptive Inference

Training. To enable adaptive inference, we supervise predictions throughout every iteration of the visual loop using the same final-answer target. Given an image–question–answer triple (x, q, y) and a maximum training depth T_{train}, the objective is

$$\mathcal{L}^{(v)} = -\sum_{t=1}^{T_{\text{train}}} \frac{w_t}{n_y} \log p_G\big(y \mid \Pi_t^{(v)}\big), \tag{3}$$

where $n_y = |y|$, $w_t > 0$ and $\sum_t w_t = 1$. We assign larger weights to later depths and compute the loss only on target-answer tokens using teacher forcing. For Variant C, the drafts in the prefix are generated by the model itself. All variants therefore require only final-answer annotations, without intermediate visual or reasoning targets.

By default, we use deep supervision (Lee et al., 2015), evaluating all depth losses for each example. Alternatively, depth-sampled supervision selects a depth with probability w_t and computes its answer-token loss. This gives an unbiased estimate of the objective at lower cost per update.

Inference. We use a simple answer-stability rule for adaptive inference. After each visual iteration, the model greedily generates an answer. We stop when two consecutive answers agree or the budget T_{test} is exhausted and return the current answer in either case. This rule requires intermediate answer generation for all three variants. AdaLoop also supports fixed-depth inference that runs for T_{test}

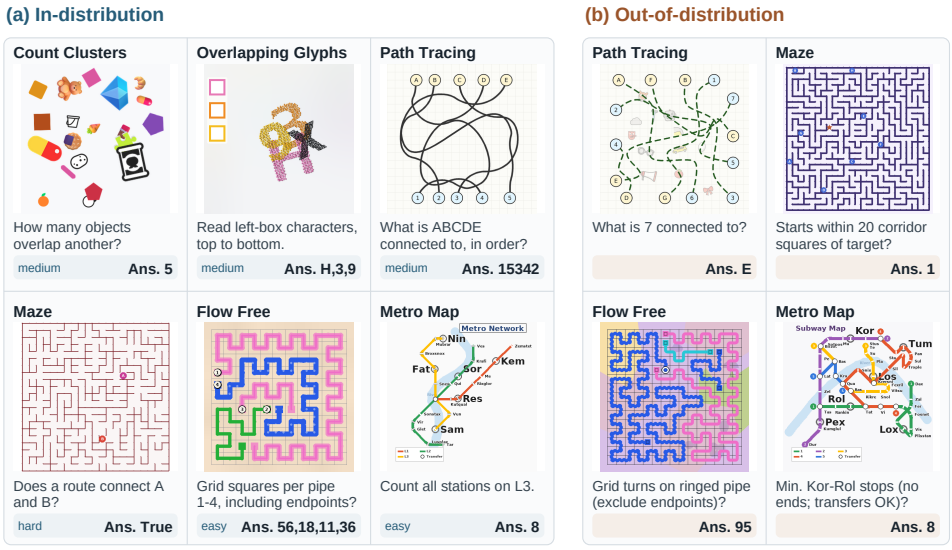


Figure 3: **VisRO examples.** (a) Examples from six core tasks, with level of difficulty shown in blue. (b) Examples from the held-out out-of-distribution (OOD) split that exceed training difficulty (e.g., with larger grids) or use held-out generation rules or appearances. Each panel shows a 448×448 image, abbreviated question, and programmatically computed answer. Appendix E has more details.

iterations. In this setting, Variant A and B generate an answer only at the final depth, while Variant C also generates intermediate drafts. In both settings, T_{test} can differ from T_{train}. Appendix B.3 provides the details of our inference procedure.

4 VisRO: A Dataset for Vision-Centric Reasoning

To study when iterative visual computation helps, we introduce **VisRO** (Visual Routines), a procedural dataset and generation pipeline for training and evaluating vision-centric reasoning. Inspired by children's visual puzzles and cognitive tests, the collection covers grouping, counting, connectivity, tracing, and recognition under occlusion, with minimal reliance on language priors or external knowledge. We use six tasks from this collection, denoted **VisRO-Core6**, for our controlled experiments. Figure 3 shows these tasks and representative out-of-distribution (OOD) examples.

Controlled difficulty and distribution shifts. Each core task provides controls over scene complexity, such as object counts, path lengths, overlap, and crossings. We use four difficulty levels for training and in-distribution evaluation. OOD tests include harder instances and shifts in scene generation rules or visual appearance. All VisRO-Core6 images are rendered at 448×448. With fixed preprocessing, this lets us vary scene complexity while keeping the per-image visual token budget constant within each backbone.

Generation and evaluation. Each generator renders a structured scene and computes ground-truth answers from its underlying representation. Answers are evaluated using task-specific, programmatic scoring rules, without an LLM judge. The underlying scenes can also support spatial supervision, such as object locations and paths through the scene. Our AdaLoop experiments use only image–question–answer triples. Appendix E describes the broader task collection and task-specific generation procedures.

5 Experiments

We implement AdaLoop on three pretrained VLMs that differ in both vision encoder and language backbone: Qwen2.5-VL (Bai et al., 2025), PaliGemma 2 (Steiner et al., 2024), and LLaVA-1.5 (Liu et al., 2024a). Our experiments are designed to answer four questions: (Sec. 5.1) Does AdaLoop consistently improve visual reasoning compared to the non-recurrent base model? (Sec. 5.2) Which design choices are critical for effective visual recurrence? (Sec. 5.3) When do additional recurrent iterations help at inference time? (Sec. 5.4) Under approximately matched training compute, how does recurrence compare with non-recurrent alternatives?

Table 1: **Results on VisRO-Core6.** Single-pass question-conditioned AdaLN improves over supervised fine-tuning (SFT), and recurrence (AdaLoop) further improves both in-distribution (ID) and out-of-distribution (OOD) accuracy across all backbones. AdaLoop uses Variant A with adaptive answer-stability stopping ($p = 1$, $T_{\text{test}} = 3$). Scores are exact-match accuracy (%); *Overall* is the macro average over six tasks. *Params*: trainable parameters (M); $\bar{T}$: mean selected stopping depth on the ID set. Closed-source models use maximum reasoning effort on a stratified 10% test subset; *tok.*: average generated tokens per completed response, including reasoning tokens. Best overall scores for each backbone are in bold.

Method	Params	$\bar{T}$ / tok.	ID							OOD
			Count	Flow	Glyph	Metro	Maze	Path	**Overall**	**Overall**
Closed-source models (zero-shot)										
GPT-6 Astra (max)	–	2,892	88.3	95.0	55.0	91.7	78.3	85.0	82.2	71.4
Claude Opus 5 (max)	–	15,807	45.0	5.0	1.7	18.3	25.0	36.7	21.9	11.7
Qwen2.5-VL-3B										
Zero-shot	0	1.0	7.3	0.0	23.5	0.0	37.3	14.8	13.8	8.9
SFT	66	1.0	56.8	12.0	56.0	59.3	56.2	59.3	49.9	23.9
Single-pass AdaLN	82	1.0	61.5	23.8	84.8	60.7	56.7	72.3	60.0	30.2
AdaLoop	82	2.2	68.5	28.0	89.2	64.0	61.8	75.8	**64.6**	**32.4**
PaliGemma2-3B										
Zero-shot	0	1.0	10.7	0.3	1.2	13.3	32.2	16.2	12.3	8.8
SFT	24	1.0	74.3	26.7	80.8	67.8	70.8	55.0	62.6	31.5
Single-pass AdaLN	39	1.0	83.2	36.8	92.7	72.7	87.8	61.0	72.4	40.0
AdaLoop	39	2.1	84.7	40.3	93.7	75.8	93.2	63.3	**75.2**	**43.9**
LLaVA-1.5-7B										
Zero-shot	0	1.0	6.0	0.2	0.5	13.7	29.5	12.7	10.4	8.6
SFT	341	1.0	48.3	18.0	62.8	51.7	59.5	62.3	50.4	24.8
Single-pass AdaLN	350	1.0	52.5	29.5	82.0	53.0	63.5	65.7	57.7	29.6
AdaLoop	350	2.2	56.8	38.7	84.7	59.0	64.2	67.3	**61.8**	**31.1**

Data and evaluation. We use the six tasks in VisRO-Core6 (Figure 3), with 600k training examples and 3,600 test examples each for in-distribution and OOD evaluation. Each split contains equal numbers of examples from all six tasks. To help retain general visual instruction-following capabilities, we mix the training data with a fixed 100k-example subset of LLaVA-NeXT (Liu et al., 2024b), using its original answer annotations. We report per-task exact-match accuracy and the macro average across tasks, separately for in-distribution and OOD evaluation.

Training and inference. Within each backbone, we compare three models, each adding one component to the previous. The SFT baseline trains the vision–language connector and language-model LoRA adapters (Hu et al., 2022), leaving the vision encoder unchanged. Single-pass AdaLN additionally trains the question pooler and modulation generators, and applies the modulated visual blocks only once. AdaLoop trains the same components but applies the loop body blocks repeatedly. All three models use the same training mixture and optimizer-update budget, with corresponding initializations and example order paired by seed. By default, we train AdaLoop through $T_{\text{train}} = 3$ and use answer-stability stopping with patience $p = 1$ at inference. Details are in Appendix B.

5.1 Main Results

Table 1 compares zero-shot inference, supervised fine-tuning (SFT), single-pass AdaLN, and AdaLoop across three pretrained VLMs. AdaLoop uses Variant A by default. Within each backbone, all instances use the same training data and number of optimizer updates.

Zero-shot evaluation. Zero-shot performance varies substantially across models. Qwen2.5-VL, PaliGemma 2, and LLaVA-1.5 achieve 13.8%, 12.3%, and 10.4% VisRO-ID accuracy, respectively. On a difficulty-stratified 10% test subset, GPT-6 Astra (OpenAI, 2026) and Claude Opus 5 (Anthropic, 2026) achieve 82.2% and 21.9% ID accuracy at maximum reasoning effort, averaging about 3k and 16k generated tokens per completed response, respectively.

Table 3: **Qwen2.5-VL ablations.** Notation as in Table 1 with shading indicating AdaLoop-A. (a) All interfaces improve over single-pass AdaLN, and Variant A performs best. (b) With no modulation, looping gains little over a single pass; modulation with question conditioning performs better at the same parameter count.

Method	Params	ID	OOD
Single-pass AdaLN	82	60.0	30.2
AdaLoop-A	82	64.6	32.4
AdaLoop-B	82	62.1	31.1
AdaLoop-C	82	63.0	32.0

(a) **Language interfaces.** A: current state; B: visual history; C: visual history and answer drafts (Fig. 2).

Modulation	Params	Single pass		Loop	
		ID	OOD	ID	OOD
None	66	49.9	23.9	50.9	23.8
AdaLN (constant)	82	57.8	29.1	60.9	30.0
AdaLN (question)	82	60.0	30.2	64.6	32.4

(b) **Modulation and recurrence.** *Constant* uses a fixed null question. Looping uses variant A.

Adapting and reusing visual computation. Adding question-conditioned AdaLN improves both ID and OOD accuracy across all three backbones, establishing a strong single-pass baseline. AdaLoop further improves both scores by repeatedly applying the conditioned visual blocks. Relative to single-pass AdaLN, ID accuracy increases by 4.6 percentage points on Qwen2.5-VL, 2.8 on PaliGemma 2, and 4.1 on LLaVA-1.5. These gains are obtained with frozen vision weights and Variant A, which exposes only the current visual state to the language model. At fixed T_{test}, AdaLoop-A also improves over single-pass AdaLN across all three backbones (Appendix C.1), without requiring intermediate answer generation for adaptive stopping.

Comparison with existing methods. Table 2 compares AdaLoop with Qwen implementations of SFT+GRPO (Shao et al., 2024b) and LIVR (Li et al., 2026b), both using final-answer supervision and matched or more training iterations. Relative to SFT, GRPO improves OOD accuracy by 0.3 points but reduces ID accuracy, while LIVR falls below SFT on both splits. AdaLoop improves both scores and outperforms these baselines under the evaluated configurations. Appendix B.4 details the baseline implementations.

Table 2: **Comparison with existing methods on Qwen2.5-VL.** ID/OOD report accuracy (%) over VisRO-Core6.

Method	Params	ID	OOD
SFT	66	49.9	23.9
+ GRPO	66	49.5	24.2
LIVR	66	47.8	22.0
AdaLoop	82	64.6	32.4

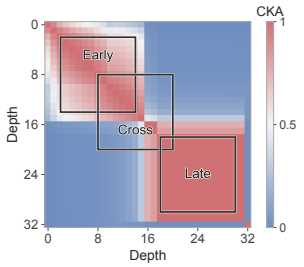
5.2 Designing useful visual recurrence

We study the language interface, visual modulation, and loop placement in AdaLoop. The ablations use Qwen2.5-VL with frozen vision weights, with training data and optimizer updates matched within each comparison. We use Variant A except when comparing language interfaces.

Language interfaces. All three interfaces improve both ID and OOD accuracy over single-pass AdaLN, and reading only the current state (A) performs best (Table 3a), so explicitly retaining earlier visual states or answer drafts in the language context is not required to benefit from visual recurrence. Retaining them does help beyond the training depth, however: B and C degrade less than A when run for more iterations (Figure 5b). One possible explanation is that the retained history lets the language model draw on earlier visual states, which are unaffected when later iterations drift beyond the trained regime.

Modulation and recurrence. Table 3b separates the contributions of modulation, question conditioning, and recurrence. Without modulation, looping raises ID accuracy only from 49.9% to 50.9%. Adding question-independent AdaLN improves the loop to 60.9%, partly through its 16M additional trainable parameters. Conditioning the same modules on the question further raises accuracy to 64.6%, without adding parameters. With question conditioning held fixed, looping improves over single-pass AdaLN by 4.6 ID and 2.2 OOD percentage points. Thus, question information and recurrent computation each contribute beyond the addition of trainable modulation parameters.

Loop placement. Figure 4a shows pretrained visual representation similarity measured by centered kernel alignment (CKA) (Kornblith et al., 2019). On Qwen2.5-VL, we compare three equal-length spans: early $[2, 14)$, cross-band $[8, 20)$, and late $[18, 30)$ (Figure 4b). All three improve overall ID accuracy over their corresponding single-pass AdaLN controls. The early span gives the largest overall relative gain and leads on four tasks, while the cross-band and late spans give the largest relative gains on Maze and Path, respectively. The benefits of recurrence therefore depend on both placement and task; useful recurrence also occurs in a span crossing the transition between high-CKA bands. CKA measurement details are provided in Appendix B.1.



(a) Encoder representation similarity. Linear CKA across depths k, and three candidate spans.

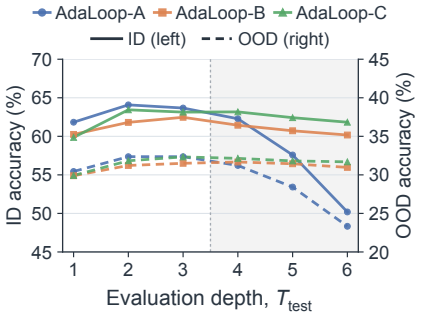
Loop span	Count	Flow	Glyph	Metro	Maze	Path	Overall
Early $[2, 14)$	**11.38**	**17.48**	**5.11**	**5.49**	9.12	4.84	**7.64**
Cross $[8, 20)$	4.52	8.23	3.87	1.63	**22.70**	−0.47	6.05
Late $[18, 30)$	6.70	2.63	2.28	2.58	−0.56	**5.57**	3.33

(b) Gains from recurrence. For each loop span $[k_{\text{start}}, k_{\text{end}})$, relative VisRO-ID accuracy gains (%) over a single-pass AdaLN, under answer-stability halting.

Figure 4: **Qwen2.5-VL loop placement.** (a) Three candidate loop spans that are contained within, or that cross, encoder spans having high CKA similarity. (b) Recurrence improves overall accuracy for all spans. Gains vary by task for each loop span.

Difficulty	Qwen2.5-VL		PaliGemma 2		LLaVA-1.5	
	Rel. gain	$\bar{T}$	Rel. gain	$\bar{T}$	Rel. gain	$\bar{T}$
Easy	+3.33	2.11	+0.87	2.06	+3.05	2.11
Medium	+5.87	2.17	+0.00	2.09	−0.17	2.15
Hard	+4.09	2.21	+1.93	2.14	+4.87	2.21
XHard	+4.46	2.26	+5.43	2.21	+7.85	2.28
OOD	+6.39	2.30	+4.49	2.23	+6.57	2.29

(a) Adaptive computation. Relative accuracy gain (%) of answer-stability stopping over one visual iteration of the same checkpoint, and mean stopping depth $\bar{T}$.



(b) Depth extrapolation. ID accuracy (solid, left axis) and OOD accuracy (dashed, right axis) on Qwen2.5-VL. Shading marks $T_{\text{test}} > T_{\text{train}}$.

Figure 5: **Inference-time visual computation.** All models are trained with $T_{\text{train}} = 3$. (a) Mean stopping depth increases with difficulty on every backbone. (b) Beyond the training depth, Variants B and C remain largely stable, whereas A degrades.

5.3 Inference-time Visual Computation

Recurrence and deep supervision enable AdaLoop to use adaptive computation at inference time. We examine how this benefit varies with problem difficulty and whether useful updates continue beyond the depths seen during training.

Adaptive computation within trained depths. Using the same checkpoint ($T_{\text{train}} = 3$), we compare adaptive stopping ($p = 1$, $T_{\text{test}} = 3$) with stopping after the first iteration. As shown in Figure 5a, adaptive stopping improves accuracy in most difficulty groups across all three backbones. Its mean stopping depth also increases with difficulty on every backbone, although neither the model nor the stopping rule sees difficulty labels. This shows that adaptive stopping using answer stability allocates more iterations to harder inputs.

Behavior beyond trained depths. Figure 5b shows that Variants B and C largely maintain accuracy beyond $T_{\text{train}} = 3$, whereas A degrades more as depth increases. Although average accuracy stays roughly flat, this stability leaves room for later iterations to correct individual examples: Figure 6 (Variant C) illustrates such cases, where predictions on Glyphs and Metro Map become correct at depth four and remain correct through depth six.

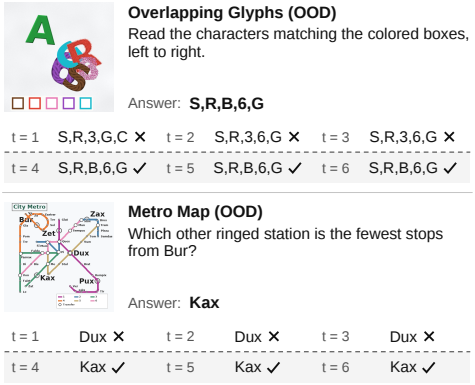
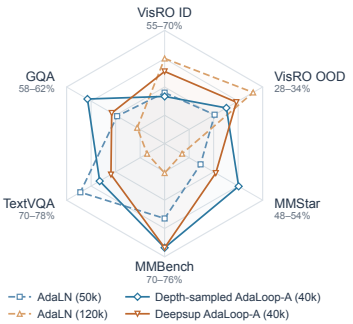


Figure 6: **Selected answer trajectories beyond the training depth** (Qwen2.5-VL, Variant C). At each depth t, the model answers from the visual history and its earlier drafts. In these selected examples, the shaded iterations ($t > T_{\text{train}}$) correct the answer and retain it.



(a) Performance profiles. Colors pair approximately matched training budgets; axes ranges differ as labeled.

Method	Train FLOPs	VisRO (ID+OOD)	MMStar	MMBench	GQA	TextVQA
AdaLN (40k)	1.00×	45.10	51.07	75.43	60.28	76.28
AdaLN (50k)	1.25×	46.40	50.20	73.97	59.94	76.88
AdaLoop (sampled, 40k)	1.32×	46.51	52.53	75.52	61.15	75.30
AdaLN (120k)	3.00×	49.85	49.07	71.56	59.12	71.43
AdaLoop (deep sup., 40k)	2.86×	48.47	51.13	75.52	60.16	74.38

(b) Training cost and accuracy. *Sampled* supervises one sampled depth per example; *deep sup.* supervises all depths. Training FLOPs are relative to single-pass AdaLN at 40k updates (Appendix C.2). VisRO averages ID and OOD accuracy (%).

Figure 7: **Qwen2.5-VL training compute allocation.** (a) Radar visualization of numbers in Table (b). At approximately matched training FLOPs, depth-sampled AdaLoop matches the 50k-update single-pass control on VisRO and improves three of four VQA benchmarks. Deeply supervised AdaLoop is slightly below the 120k-update control on VisRO but outperforms it on all four general VQA benchmarks, using one third as many optimizer updates.

5.4 Computational Trade-offs

AdaLoop achieves better performance at matched optimizer updates (Sec. 5.1), but looping incurs higher training and inference costs than a parameter-matched single-pass model. We quantify these costs on Qwen2.5-VL and compare models under approximately matched training compute.

Inference cost. On VisRO-ID and with fixed-depth inference at $T_{\text{test}} = 3$, AdaLoop generates an answer only after the final visual iteration, using an estimated $1.3\times$ the inference FLOPs of single-pass AdaLN. Answer-stability stopping with the same depth limit has an estimated cost of $2.2\times$ on average because it also generates intermediate answers. Appendix C.2 describes the accounting.

Training cost. At $T_{\text{train}} = 3$, deep supervision at every depth costs approximately $2.9\times$ the training FLOPs per update of single-pass AdaLN. Sampling one supervised depth according to the normalized loss weights reduces the estimated cost to approximately $1.3\times$, while preserving the weighted objective in expectation. We therefore compare depth-sampled and deeply supervised AdaLoop, each trained for 40k updates, with single-pass AdaLN trained for 50k and 120k updates, respectively; total training FLOPs match to within 6% (Figure 7b). AdaLoop thus uses 20% and 67% fewer optimizer updates, and correspondingly fewer passes over the training data.

Performance at approximately matched training compute. Besides VisRO, we also evaluate on GQA (Hudson & Manning, 2019), MMBench (Liu et al., 2024c), MMStar (Chen et al., 2024), and TextVQA (Singh et al., 2019). These benchmarks measure retention of general VQA capabilities acquired by the backbones during their original training, and our post-training mixture includes a LLaVA-NeXT subset to limit forgetting (Section 5).

At the lower budget (Figure 7), depth-sampled AdaLoop matches the 50k-update control on VisRO and is higher on three general VQA benchmarks. At the higher budget, deeply supervised AdaLoop is slightly below the 120k-update control on VisRO but outperforms it on all four VQA benchmarks. Extending single-pass AdaLN training from 40k to 120k updates lowers all general benchmarks (e.g., MMBench from 75.4% to 71.6%), consistent with forgetting. These results suggest that recurrence offers a different way to spend training compute: more computation per example rather than more updates over the same data. At approximately matched FLOPs, AdaLoop achieves VisRO accuracy competitive with extended single-pass training while retaining better general capabilities.

6 Conclusion

We introduced AdaLoop, a framework for question-conditioned visual recurrence, and VisRO, a procedural dataset and generation pipeline for vision-centric reasoning. Across three VLM families, question conditioning and recurrence provide complementary gains. With training data and optimizer updates matched, AdaLoop improves over both supervised fine-tuning and question-conditioned single-pass controls. These gains are obtained with the pretrained vision weights frozen. Our findings suggest that a pretrained vision encoder need not be a fixed front end: its existing visual transformations can be adapted and reused for iterative reasoning. We hope this work encourages further study of how vision modules can contribute to multimodal reasoning.

AI USE STATEMENT

In this work, the authors designed the core algorithm, pseudocode and overall method and dataset framework. They developed the paper's central narrative and drafted the manuscript. Generative AI tools assisted with method implementation, training infrastructure, dataset loaders, and procedural data-generation code. VisRO examples and their ground-truth answers were generated programmatically from structured scenes. AI tools also provided feedback on research hypotheses, experimental design, and interpretation of results. Additional assistance included literature survey, manuscript editing, figure preparation, and reference formatting. The authors reviewed all AI-assisted contributions and tested AI-assisted code. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We will release our code, model checkpoints, experiment configurations, VisRO generators, and fixed training and test datasets. The release will include data-split definitions and identifiers for the LLaVA-NeXT subset used in training. Appendix B describes the implementation, training, and inference procedures, while Appendix E documents data generation and evaluation. Appendix A states the assumptions and approximations underlying our analysis of AdaLN and recurrence.

REFERENCES

Ofir Abramovich, Niv Nayman, Sharon Fogel, Inbal Lavi, Ron Litman, Shahar Tsiper, Royee Tichauer, Srikar Appalaraju, Shai Mazor, and R. Manmatha. VisFocus: Prompt-guided vision encoders for OCR-free dense document understanding. In *ECCV*, 2024.

Ibrahim Alabdulmohsin and Xiaohua Zhai. Recursive inference scaling: A winning path to scalable inference in language and multimodal systems. In *NeurIPS*, 2025.

Ibrahim Alabdulmohsin, Xiaohua Zhai, Alexander Kolesnikov, and Lucas Beyer. Getting ViT in shape: Scaling laws for compute-optimal model design. In *NeurIPS*, 2023.

Anthropic. Introducing Claude Opus 5, 2026. URL `https://www.anthropic.com/news/claude-opus-5`.

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise LoRA. In *ICLR*, 2025a.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. In *NeurIPS*, 2025b.

Junyeob Baek, Mingyu Jo, Minsu Kim, Mengye Ren, Yoshua Bengio, and Sungjin Ahn. Generative recursive reasoning. *arXiv preprint arXiv:2605.19376*, 2026.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-VL technical report. *arXiv preprint arXiv:2502.13923*, 2025.

Daniel Bolya, Po-Yao Huang, Peize Sun, Jang Hyun Cho, Andrea Madotto, Chen Wei, Tengyu Ma, Jiale Zhi, Jathushan Rajasegaran, Hanoona Bangalath, Junke Wang, Marco Monteiro, Hu Xu, Shiyu Dong, Nikhila Ravi, Shang-Wen Li, Piotr Dollár, and Christoph Feichtenhofer. Perception encoder: The best visual embeddings are not at the output of the network. In *NeurIPS*, 2025.

Liang Chen, Weichu Xie, Yiyan Liang, Hongfeng He, Hans Zhao, Zhibo Yang, Zhiqi Huang, Haoning Wu, Haoyu Lu, Y. Charles, Yiping Bao, Yuantao Fan, Guopeng Li, Haiyang Shen, Xuanzhong Chen, Wendong Xu, Shuzheng Si, Zefan Cai, Wenhao Chai, Ziqi Huang, Fangfu Liu, Tianyu Liu, Baobao Chang, Ming Wu, Xiaobo Hu, Kaiyuan Chen, Yixin Ren, Yang Liu, Yuan Gong, and Kuan Li. BabyVision: Visual reasoning beyond language. In *ICML*, 2026a.

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*, 2024.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. Neural ordinary differential equations. In *NeurIPS*, 2018.

Zixi Chen, Akshay Vegesna, Samip Dahal, and Andrew Gordon Wilson. How model growth, recursion, and boundary operators influence scaling exponents. *arXiv preprint arXiv:2609.19107*, 2026b.

Wenliang Dai, Junnan Li, Dongxu Li, Anthony Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale N. Fung, and Steven Hoi. InstructBLIP: Towards general-purpose vision-language models with instruction tuning. In *NeurIPS*, 2023.

Harm de Vries, Florian Strub, Jérémie Mary, Hugo Larochelle, Olivier Pietquin, and Aaron C. Courville. Modulating early visual processing by language. In *NeurIPS*, 2017.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal Transformers. In *ICLR*, 2019.

Xingyu Fu, Yushi Hu, Bangzheng Li, Yu Feng, Haoyu Wang, Xudong Lin, Dan Roth, Noah A. Smith, Wei-Chiu Ma, and Ranjay Krishna. BLINK: Multimodal large language models can see but not perceive. In *ECCV*, 2024.

Alessio Galatolo, Ronald Cumbal, Alexandros Rouchitsas, Katie Winkle, Didem Gürdür Broo, and Ginevra Castellano. Lightweight visual reasoning for socially-aware robots. In *ICRA*, 2026.

Roy Ganz, Yair Kittenplon, Aviad Aberdam, Elad Ben Avraham, Oren Nuriel, Shai Mazor, and Ron Litman. Question aware vision transformer for multimodal reasoning. In *CVPR*, 2024.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *NeurIPS*, 2025.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *ICML*, 2023.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

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

Keya Hu, Ali Cy, Linlu Qiu, Xiaoman Delores Ding, Runqian Wang, Yeyin Eva Zhu, Jacob Andreas, and Kaiming He. ARC is a vision problem! In *CVPR*, 2026.

Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Ranjay Krishna. Visual Sketchpad: Sketching as a visual chain of thought for multimodal language models. In *NeurIPS*, 2024.

Benhao Huang, Chufan Shi, Junlin Chen, Shicheng Wen, Zhengzhong Liu, Eric Xing, and Xuezhe Ma. Towards looped models done right—Part I: Topology, input injection, recurrent-state design, 2026. URL `https://ifm-research.notion.site/Towards-Loop-Models-Done-Right-3ade511912ec8128987dfeb7a5580043`. Accessed 2026-07-31.

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

Mozes Jacobs, Thomas Fel, Richard Hakim, Alessandra Brondetta, Demba Ba, and T. Andy Keller. Block-recurrent dynamics in ViTs. In *ICLR*, 2026.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. LoopFormer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. In *ICLR*, 2026.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*, 2017.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv preprint arXiv:2510.04871*, 2025.

Jongha Kim, Minseong Bae, Sanghyeok Lee, Jinsung Yoon, and Hyunwoo J. Kim. TabFlash: Efficient table understanding with progressive question conditioning and token focusing. In *AAAI*, 2026.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *ICML*, 2019.

Victor A. F. Lamme and Pieter R. Roelfsema. The distinct modes of vision offered by feedforward and recurrent processing. *Trends in Neurosciences*, 2000.

Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *AISTATS*, 2015.

Ryan Lee, Jacob Biloki, Edward J. Hu, and Jonathan May. Sparse layers are critical to scaling looped language models. *arXiv preprint arXiv:2605.09165*, 2026.

Sang In Lee and Jihun Park. Soft Mixture-of-Recursions: Going deeper with recursive vision transformers. *arXiv preprint arXiv:2607.00774*, 2026.

Bangzheng Li, Ximeng Sun, Jiang Liu, Ze Wang, Jialian Wu, Xiaodong Yu, Hao Chen, Emad Barsoum, Muhao Chen, and Zicheng Liu. Latent visual reasoning. In *ICLR*, 2026a.

Kelvin Li, Chuyi Shang, Leonid Karlinsky, Rogerio Feris, Trevor Darrell, and Roei Herzig. Latent implicit visual reasoning. In *CVPR*, 2026b.

Ming Liang and Xiaolin Hu. Recurrent convolutional neural network for object recognition. In *CVPR*, 2015.

Qianli Liao and Tomaso Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *arXiv preprint arXiv:1604.03640*, 2016.

Drew Linsley, Junkyung Kim, Vijay Veerabadran, Charles Windolf, and Thomas Serre. Learning long-range spatial dependencies with horizontal gated recurrent units. In *NeurIPS*, 2018.

Hanpeng Liu, Yaqian Li, Zidan Wang, Shuoxi Zhang, Zihao Bo, Rinyoichi Takezoe, Kaiwen Long, and Kun He. iGVLM: Dynamic instruction-guided vision encoding for question-aware multimodal understanding. *arXiv preprint arXiv:2603.02748*, 2026a.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. In *CVPR*, 2024a.

Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. LLaVA-NeXT: Improved reasoning, OCR, and world knowledge, 2024b. URL `https://llava-vl.github.io/blog/2024-01-30-llava-next/`.

Ying Liu, Yudong Han, Kean Shi, and Liyuan Pan. Mema: Memory-augmented adapter for enhanced vision-language understanding. *arXiv preprint arXiv:2603.00655*, 2026b.

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, Kai Chen, and Dahua Lin. MMBench: Is your multi-modal model an all-around player? In *ECCV*, 2024c.

Chengzhi Mao, Xudong Lin, and Wen-Sheng Chu. Language-instructed vision embeddings for controllable and generalizable perception. In *ICLR*, 2026.

Sean McLeish, Ang Li, John Kirchenbauer, Dayal Singh Kalra, Brian R. Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Jonas Geiping, Tom Goldstein, and Micah Goldblum. Teaching pretrained language models to think deeper with retrofitted recurrence. In *COLM*, 2026.

OpenAI. GPT-6 Astra system card, 2026. URL https://deploymentsafety.openai.com/gpt-6-astra.

Suhyeong Park, Junha Jung, and Jaewoo Kang. Reason through the latent! Making latent visual reasoning necessary. *arXiv preprint arXiv:2609.06746*, 2026a.

Taekhyun Park, Yongjae Lee, Dohee Kim, and Hyerim Bae. LoopUS: Recasting pretrained LLMs into looped latent refinement models. *arXiv preprint arXiv:2605.11011*, 2026b.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICCV*, 2023.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *AAAI*, 2018.

Pedro Pinheiro and Ronan Collobert. Recurrent convolutional neural networks for scene labeling. In *ICML*, 2014.

Hayden Prairie, Zachary Novack, Taylor Berg-Kirkpatrick, and Daniel Y Fu. Parcae: Scaling laws for stable looped language models. In *COLM*, 2026.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, 2021.

Rajesh P. N. Rao and Dana H. Ballard. Predictive coding in the visual cortex: A functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 1999.

Kristian Schwethelm, Daniel Rueckert, and Georgios Kaissis. How much is one recurrence worth? Iso-depth scaling laws for looped language models. *arXiv preprint arXiv:2604.21106*, 2026.

Hao Shao, Shengju Qian, Han Xiao, Guanglu Song, Zhuofan Zong, Letian Wang, Yu Liu, and Hongsheng Li. Visual CoT: Advancing multi-modal language models with a comprehensive dataset and benchmark for chain-of-thought reasoning. In *NeurIPS Datasets and Benchmarks Track*, 2024a.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024b.

Haozhan Shen, Kangjia Zhao, Tiancheng Zhao, Ruochen Xu, Zilun Zhang, Mingwei Zhu, and Jianwei Yin. ZoomEye: Enhancing multimodal LLMs with human-like zooming capabilities through tree-based image exploration. In *EMNLP*, 2025.

Wen-Jie Shu, Xuerui Qiu, Rui-Jie Zhu, Harold Haodong Chen, Yexin Liu, and Harry Yang. LoopViT: Scaling visual ARC with looped transformers. *arXiv preprint arXiv:2602.02156*, 2026.

Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. Towards VQA models that can read. In *CVPR*, 2019.

Andreas Steiner, André Susano Pinto, Michael Tschannen, Daniel Keysers, Xiao Wang, Yonatan Bitton, Alexey Gritsenko, Matthias Minderer, Anthony Sherbondy, Shangbang Long, Siyang Qin, Reeve Ingle, Emanuele Bugliarello, Sahar Kazemzadeh, Thomas Mesnard, Ibrahim Alabdulmohsin, Lucas Beyer, and Xiaohua Zhai. PaliGemma 2: A family of versatile VLMs for transfer. *arXiv preprint arXiv:2412.03555*, 2024.

Shengbang Tong, Ellis Brown, Penghao Wu, Sanghyun Woo, Manoj Middepogu, Sai C. Akula, Jihan Yang, Shusheng Yang, Adithya Iyer, Xichen Pan, Austin Wang, Rob Fergus, Yann LeCun, and Saining Xie. Cambrian-1: A fully open, vision-centric exploration of multimodal LLMs. In *NeurIPS*, 2024a.

Shengbang Tong, Zhuang Liu, Yuexiang Zhai, Yi Ma, Yann LeCun, and Saining Xie. Eyes wide shut? Exploring the visual shortcomings of multimodal LLMs. In *CVPR*, 2024b.

Shimon Ullman. Visual routines. *Cognition*, 1984.

André G. Viveiros, Nuno Gonçalves, Matthias Lindemann, and André F. T. Martins. LanteRn: Latent visual structured reasoning. In *ICLR Workshop on Multimodal Intelligence*, 2026.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025.

Maijunxian Wang, Ruisi Wang, Juyi Lin, Ran Ji, Thaddäus Wiedemer, Qingying Gao, Dezhi Luo, Yaoyao Qian, Lianyu Huang, Zelong Hong, Jiahui Ge, Qianli Ma, Hang He, Yifan Zhou, Lingzi Guo, Lantao Mei, Jiachen Li, Hanwen Xing, Tianqi Zhao, Fengyuan Yu, Weihang Xiao, Yizheng Jiao, Jianheng Hou, Danyang Zhang, Pengcheng Xu, Boyang Zhong, Zehong Zhao, Gaoyun Fang, John Kitaoka, Yile Xu, Hua Xu, Kenton Blacutt, Tin Nguyen, Siyuan Song, Haoran Sun, Shaoyue Wen, Linyang He, Runming Wang, Yanzhi Wang, Mengyue Yang, Ziqiao Ma, Raphaël Millière, Freda Shi, Nuno Vasconcelos, Daniel Khashabi, Alan Yuille, Yilun Du, Ziming Liu, Dahua Lin, Ziwei Liu, Vikash Kumar, Yijiang Li, Lei Yang, Zhongang Cai, and Hokin Deng. A very big video reasoning suite. In *ICML*, 2026a.

Qixun Wang, Yang Shi, Yifei Wang, Yuanxing Zhang, Pengfei Wan, Kun Gai, Xianghua Ying, and Yisen Wang. Monet: Reasoning in latent visual space beyond image and language. In *CVPR*, 2026b.

Shaowen Wang, Ge Zhang, Kairong Luo, Yuhao Wu, Shaofan Liu, Jiaheng Liu, Wenhao Huang, Shen Yan, and Jian Li. SMELT: Scaling laws for compute-matched MoE looped transformers. *arXiv preprint arXiv:2609.01343*, 2026c.

Penghao Wu and Saining Xie. V*: Guided visual search as a core mechanism in multimodal LLMs. In *CVPR*, 2024.

Junxiang Xu, Ruisi Wang, Fanyi Pu, Maijunxian Wang, Ran Ji, Tongxi Zhou, Chenyang Gu, Jing Zuo, Hongcan Xiao, Yimeng Geng, Wanqi Yin, Wei Chen, Oscar Qian, Zhengan Yan, Ziqi Huang, Haiwen Diao, Liang Pan, Bo Li, Xiangyu Fan, Dezhi Luo, Fengyuan Yu, Zehong Zhao, Qingying Gao, Tinghui Zhu, Yilan Zhang, Jingqi Tong, Pinyuan Feng, Zhengze Jiang, Letian Wang, Ziyu Guo, Renrui Zhang, Jieneng Chen, Sonia Joseph, Constantin Venhoff, Saman Motamed, Mengyue Yang, Chandra Sripada, Alan Yuille, Philip Torr, Lvmin Zhang, Vikash Kumar, Daniel Khashabi, Nikolaus Kriegeskorte, Raphaël Millière, Vincent C. Müller, Anyi Rao, Quan Wang, Ziwei Liu, Dahua Lin, Lei Yang, Hokin Deng, and Zhongang Cai. VBVR-Pro: A scalable and verifiable suite for native visual reasoning. *arXiv preprint arXiv:2608.26105*, 2026a.

Ruihan Xu, Yuting Gao, Lan Wang, Jianing Li, Weihao Chen, Qingpei Guo, Ming Yang, and Shiliang Zhang. Looping back to move forward: Recursive transformers for efficient and flexible large multimodal models. *arXiv preprint arXiv:2602.09080*, 2026b.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-Mark prompting unleashes extraordinary visual grounding in GPT-4V. *arXiv preprint arXiv:2310.11441*, 2023.

Runpeng Yu, Xinyin Ma, and Xinchao Wang. Auto-controlled image perception in MLLMs via visual perception tokens. In *ICCV*, 2025.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *ICCV*, 2023.

Yiming Zhang, Qiangyu Yan, Borui Jiang, and Kai Han. Multimodal latent reasoning via hierarchical visual cues injection. *arXiv preprint arXiv:2602.05359*, 2026a.

Yuyao Zhang, Zefeng Chen, Haizhai Zhang, Guorui Chen, Chang Yang, and Zhijiang Li. DVR: Dynamic visual re-reading for multimodal large language models via sub-question feedback. *Neurocomputing*, 700:134448, 2026b.

Yaowei Zheng, Junting Lu, Shenzhi Wang, Zhangchi Feng, Dongdong Kuang, Yuwen Xiong, and Richong Zhang. EasyR1: An efficient, scalable, multi-modality RL training framework. `https://github.com/hiyouga/EasyR1`, 2025.

Ziwei Zheng, Michael Yang, Jack Hong, Chenxiao Zhao, Guohai Xu, Le Yang, Chao Shen, and Xing Yu. DeepEyes: Incentivizing "thinking with images" via reinforcement learning. In *ICLR*, 2026.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. BERT loses patience: Fast and robust inference with early exit. In *NeurIPS*, 2020.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, Qiyang Min, Hongzhi Huang, Xun Zhou, Wei Ye, Jiaheng Liu, Jian Yang, Yunfeng Shi, Chenghua Lin, Enduo Zhao, Tianle Cai, Ge Zhang, Wenhao Huang, Yoshua Bengio, and Jason Eshraghian. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

A AdaLN as Task-Conditioned Visual Compute Allocation

This section gives an intuitive account of why question-conditioned adaptive layer normalization (AdaLN) is useful when adapting a pretrained vision encoder. The purpose is not to establish an exact equivalence between feature modulation and recurrent computation. Rather, a local analysis shows that AdaLN provides a fine-grained, first-order mechanism for reallocating the visual transformations already present in the encoder, while recurrence provides the complementary ability to recompute those transformations at an updated visual state. Throughout, "compute allocation" refers to the relative influence of transformations already available in the encoder rather than to arithmetic: modulation changes which pretrained updates dominate without changing the FLOPs of a single pass. Recurrence is what increases executed depth.

A.1 From AdaLN to a modulated residual update

For clarity, consider one pre-normalized residual branch at layer ℓ and loop iteration t. Let $x_{\ell,t} \in \mathbb{R}^{N\times d}$ denote N visual tokens with channel dimension d, and let q denote the question representation. We write our modulation abstractly as

$$z_{\ell,t} = \mathrm{LN}(x_{\ell,t}), \tag{4}$$

$$\widetilde{z}_{\ell,t} = \big(\mathbf{1} + \gamma_\ell(q)\big) \odot z_{\ell,t} + \beta_\ell(q), \tag{5}$$

$$x^{+}_{\ell,t} = x_{\ell,t} + \big(\mathbf{1} + \alpha_\ell(q)\big) \odot F_\ell(\widetilde{z}_{\ell,t}), \tag{6}$$

where F_ℓ is the update produced by the transformer layer, and $\alpha_\ell, \beta_\ell, \gamma_\ell \in \mathbb{R}^d$ are the residual gate, shift, and scale of Eq. (2), broadcast over visual tokens. We suppress the sublayer superscript f, since the same notation applies separately to the attention and MLP sub-layers of a transformer block. Conditioning on q allows the same pretrained block to emphasize different channels for different questions. The modulation is reused across loop iterations.

Let $u^0_{\ell,t} = F_\ell(z_{\ell,t})$ be the update produced by the unmodulated sub-layer and define $\delta z_{\ell,t} = \gamma_\ell \odot z_{\ell,t} + \beta_\ell$, so that $\widetilde{z}_{\ell,t} = z_{\ell,t} + \delta z_{\ell,t}$. A first-order expansion around the pretrained computation gives

$$x^{+}_{\ell,t} = x_{\ell,t} + (\mathbf{1} + \alpha_\ell) \odot \Big(u^0_{\ell,t} + J_{F_\ell}(z_{\ell,t})\big[\gamma_\ell \odot z_{\ell,t} + \beta_\ell\big]\Big) + \mathcal{O}(\|\delta z_{\ell,t}\|^2). \tag{7}$$

Equation (7) makes the computational role of AdaLN explicit: it does not create a new visual operator from scratch, but steers the update generated by a pretrained operator using a low-dimensional, question-dependent intervention. The scale γ_ℓ is multiplicative in the current visual activity, the gate α_ℓ rescales the resulting update channel by channel, and the shift β_ℓ injects a channel-wise bias shared across visual tokens. Thus, the two multiplicative terms preserve the image- and token-dependent structure of $z_{\ell,t}$ and can selectively amplify or suppress transformations that are already active; the additive term does not. This qualitative distinction does not by itself prove that scaling must be more effective, but it is consistent with our preliminary observations, on an earlier version of our data, that the scale component accounts for most of the gain from AdaLN.

A.2 A first-order connection to effective depth

Throughout this section we write the visual state of a block as a single vector $x \in \mathbb{R}^{Nd}$, stacking its N tokens, and $F\colon \mathbb{R}^{Nd} \to \mathbb{R}^{Nd}$ for the residual branch acting on it. We allow F to mix tokens, as attention does, so its Jacobian carries both channel and cross-token dependence. A channel-wise quantity $v \in \mathbb{R}^d$ acts on the state as the matrix $\Gamma_v = I_N \otimes \mathrm{Diag}(v) \in \mathbb{R}^{Nd\times Nd}$, with Diag making the action channel-wise and $I_N\otimes$ applying it identically to every token. Such matrices compose as $\Gamma_u\Gamma_v = \Gamma_{u\odot v}$.

To isolate the scale intuition, consider a surrogate model where the question modulates the block only by rescaling its update:

$$x^{+} \;=\; x + \Gamma_g\, F(x), \tag{8}$$

with question- and depth-dependent channel-wise gain $g(q,t) \in \mathbb{R}^d$. This section and the next analyze which effects of recurrence—repeated application of F—a single gain-modulated pass can reproduce.

Equation (8) is an identity for AdaLN's gate path because the residual gate contributes $x^+ = x + \Gamma_{\mathbf{1}+\alpha} F(x)$. Whether the pre-branch scale γ and shift β can be absorbed into the same gain is less immediate. The scale reduces to a gain in a restricted case: if the branch acts on the normalized input z of (4) by a shared channel-wise scaling, $F(z) = \Gamma_\lambda z$ with $\lambda \in \mathbb{R}^d$, then $\Gamma_\lambda \Gamma_{\mathbf{1}+\gamma} = \Gamma_{\mathbf{1}+\gamma} \Gamma_\lambda$, recovering (8) with $\Gamma_g = \Gamma_{\mathbf{1}+\alpha} \Gamma_{\mathbf{1}+\gamma}$. Channel mixing, token mixing, and nonlinearities, including the attention softmax, break this commutation, so in general the scale matches a gain only in a neighborhood of the current state, and it changes as that state evolves. The shift matches no gain at all, entering by (7) as a bias injection rather than a rescaling, we omit it to isolate the effect of scaling, so the analysis below concerns the multiplicative part of AdaLN only. We use (8) as an interpretive model of the combined effect, not as an identity for the full nonlinear block.

Residual networks admit the familiar interpretation of an Euler discretization of a continuous transformation (Chen et al., 2018). Introduce a nominal step size ϵ, equal to 1 in a pretrained block, and define

$$R_\epsilon(x) = x + \epsilon F(x). \tag{9}$$

For a fixed integer n, applying this update n times and expanding around the initial state yields

$$R_\epsilon^n(x) = x + n\epsilon F(x) + \binom{n}{2} \epsilon^2 J_F(x) F(x) + \mathcal{O}_n(\epsilon^3), \tag{10}$$

where the subscript on $\mathcal{O}_n$ indicates a constant that grows with n. (The expansion says which terms recurrence contributes, not how large they are.)

By contrast, a single scalar-gated update, with $c \in \mathbb{R}$, is

$$\widehat{R}_{\epsilon,c}(x) = x + \epsilon c F(x). \tag{11}$$

Setting $c = n$ reproduces the first-order displacement in Eq. (10) exactly, so the two leading terms in that expression divide between the two mechanisms. The gain accounts for the displacement, meaning how far the representation moves along the currently available update direction, with c free to be fractional. Recurrence accounts for the second-order term $J_F(x)F(x)$, which exists because F is reevaluated after the state moves. Residual scaling therefore allocates effective depth, and recurrence supplies what follows from recomputing the direction.

The linear case makes this distinction especially transparent, because there the n-step update is exact instead of an expansion. If $F(x) = Wx$ with $W \in \mathbb{R}^{Nd \times Nd}$, then

$$R_\epsilon^n(x) = (I + \epsilon W)^n x, \qquad \widehat{R}_{\epsilon,\Gamma_g}(x) = (I + \epsilon \Gamma_g W) x. \tag{12}$$

Suppose W shares the structure of Γ_g, acting identically and diagonally on every token: $W = \Gamma_\lambda$ with $\lambda = (\lambda_1, \ldots, \lambda_d) \in \mathbb{R}^d$. A channel-wise gain can then match the n-step linear transformation exactly within this residual-gain surrogate:

$$g_i^\star(n) = \frac{(1 + \epsilon\lambda_i)^n - 1}{\epsilon \lambda_i}, \qquad g_i^\star(n) = n \text{ when } \lambda_i = 0. \tag{13}$$

For small $\epsilon\lambda_i$, this reduces to $g_i^\star(n) = n + \binom{n}{2}\epsilon\lambda_i + \cdots$, so the gain that reproduces n iterations is, to leading order, n itself. Moreover n enters Eq. (13) only as an exponent, so $g_i^\star$ is defined for non-integer n, and the gain can interpolate continuously between integer depths and hold a different effective depth in each channel. The correspondence holds only while W shares both of Γ_g's factors, however. If W varies across tokens or mixes them, as attention does, exact matching would require a gain that does the same, which the factor I_N forbids; and if W is dense in channels, the target $(I + \epsilon W)^n$ contains the mixtures $W^2, W^3, \ldots$ that a diagonal gain cannot produce.

The two obstructions just described are the linear case of a more general mismatch, which is stronger still when F is nonlinear, because the Jacobian changes with the evolving state. The leading discrepancy after scalar first-order matching ($c = n$) is already visible in Eq. (10):

$$R_\epsilon^n(x) - \widehat{R}_{\epsilon,n}(x) = \binom{n}{2} \epsilon^2 J_F(x) F(x) + \mathcal{O}_n(\epsilon^3). \tag{14}$$

A richer gain absorbs part of this term but not all of it. Allowing the gain its own first-order correction, $\Gamma_{n\mathbf{1}+\epsilon\mu}$ with $\mu \in \mathbb{R}^d$, adds a second-order term $\epsilon^2 \Gamma_\mu F(x)$. Matching $\binom{n}{2}\epsilon^2 J_F(x)F(x)$

then requires $\mu_i[F(x)]_{j,i} = \binom{n}{2}[J_F(x)F(x)]_{j,i}$ at every token j and channel i, which is Nd equations in d unknowns. A token-shared gain can therefore absorb whatever part of $J_F(x)F(x)$ is a token-uniform channel-wise rescaling of $F(x)$, but not the part that varies across tokens.

The quantity $J_F(x)F(x)$ measures how the appropriate update changes after one refinement step, which suggests a broader task-level hypothesis. Recurrence should be most useful when solving a task that requires successive updates to the internal relational organization of a scene, such as propagating reachability through a maze or forming and consolidating visual groups before counting them. In such cases, each partial assignment changes where or how the next update should act, producing computation that cannot generally be reduced to rescaling a fixed update direction. By contrast, tasks whose relevant structure is already captured by a stable feedforward representation may benefit less from additional visual recurrence. Whether a task demands iterative state construction depends on how much structure a single pass already resolves, not on the task's label.

A.3 Why AdaLN and recurrence are complementary

The analysis above separates two axes that ordinary network depth conflates. AdaLN provides *allocation*: it selects, per layer and per channel, how strongly a pretrained transformation contributes for a given question. Recurrence provides *recomputation*: it reevaluates that transformation at the state produced by the preceding iteration. They also differ in granularity: AdaLN acts per channel and continuously, by (13), whereas a loop acts per span and in discrete steps.

This view also clarifies why a uniform, unconditioned loop can be a poor way to reuse a pretrained hierarchy. Different blocks implement transformations at different levels of abstraction, different questions need not benefit from repeating all of them equally, and a single loop count imposes the same coarse allocation on every block in the recurrent span.

AdaLN's conditioning removes that constraint. The frozen blocks and the question-derived modulation are both reused at every iteration, so the loop body is a single shared operator; but because the modulation depends on q, that operator becomes a family indexed by the question:

$$x_{t+1} = \mathcal{L}\big(x_t; \{\alpha_\ell(q), \beta_\ell(q), \gamma_\ell(q)\}_{\ell\in\mathcal{B}}\big), \tag{15}$$

where $\mathcal{L}$ denotes the shared loop body and $\mathcal{B}$ its set of blocks. Training learns the map $q \mapsto \{\alpha_\ell(q), \beta_\ell(q), \gamma_\ell(q)\}$ rather than a single update rule, so different questions emphasize different feature dimensions of the same pretrained span. The updates applied at successive iterations differ because, while $\mathcal{L}$ is fixed across t, its input x_t is not.

Takeaway. AdaLN and looping are not competing approximations to the same operation. AdaLN selects and reweights the pretrained visual computation appropriate for a question, which locally resembles a channel-wise, fractional allocation of effective depth. Looping supplies what a single modulated pass cannot: repeated evaluation at the evolving visual state, including the higher-order term $J_F F$ of (14) associated with propagation and feedback. Their combination realizes task-conditioned iterative visual computation under final-answer supervision alone.

B Implementation Details

Table 4 lists the model configurations and training hyperparameters. Below, we describe the adapters, training procedure, inference protocol, and baseline implementations. Dataset generation and scoring are described in Appendix E; FLOP accounting is given in Appendix C.2.

B.1 Model and Adapter Configurations

Backbones and loop spans. We use Qwen2.5-VL-3B-Instruct, PaliGemma 2 3B Mix-448, and our re-trained LLaVA-1.5-7B checkpoint at 448×448 (Table 4). The notation $[a, b)$ denotes encoder blocks $a, \ldots, b-1$. Each backbone retains its vision–language connector after the vision encoder. Qwen's spatial patch merger reduces each visual state to one quarter as many language-model input tokens, without changing the resolution of the recurrent state. LLaVA applies 2×2 average pooling to its 1,024 CLIP patch tokens, producing 256 tokens for the language model.

Table 4: **Configurations and training hyperparameters.** Defaults for the main experiments; exceptions are specified with the corresponding results. Batch sizes are global, and training length and warmup count optimizer updates. N denotes the number of recurrent visual tokens in Qwen. Encoder references: SigLIP So400m/14 (Zhai et al., 2023; Alabdulmohsin et al., 2023), CLIP ViT-L/14 (Radford et al., 2021). [§]We use our re-trained LLaVA-1.5-7B checkpoint at 448×448, with 2×2 average pooling of the 1,024 CLIP patch tokens, rather than the released 336-px checkpoint.

Setting	Qwen2.5-VL-3B	PaliGemma 2 3B	LLaVA-1.5-7B[§]
Vision encoder	Native ViT	SigLIP So400m/14	CLIP ViT-L/14
Vision hidden width d_v	1280	1152	1024
LM embedding width	2048	2304	4096
Image preprocessing	Dynamic	448×448	448×448
Visual tokens for the LM	$N/4$	1024	256
LoRA rank	64	64	128
LoRA alpha	128	128	256
LoRA dropout	0	0	0.05
LoRA target modules	LM attn. q, k, v, o	LM attn. q, k, v, o	All LM linear
SFT trainable params (M)	66	24	341
AdaLN / AdaLoop params (M)	82	39	350
Training updates	40k	30k	40k
Batch size		64	
Optimizer		AdamW	
Adam (β_1, β_2)	$(0.9, 0.99)$	$(0.9, 0.999)$	$(0.9, 0.95)$
Adam ϵ		10^{-8}	
Learning rate: LoRA		1×10^{-4}	
Learning rate: connector	5×10^{-5}	3×10^{-5}	5×10^{-5}
Learning rate: conditioning		1×10^{-4}	
Learning-rate schedule	Cosine to $0.1\times$	Cosine to $0.1\times$	Cosine to 0
Warmup updates	200	200	1,200
Weight decay	10^{-4}	10^{-4}	0
Gradient clipping norm		1.0	
Training precision		BF16	
Default loop span	$[2, 14)$	$[3, 15)$	$[2, 8)$
Default interface		Variant A	
Training depth T_{train}		3	
Depth-weight ratio r	0.6	0.8	0.6
Depth supervision		All T_{train} depths, static geometric weights	
Modulation initialization		Zero output projection	

CKA measurement. Figure 8 shows linear CKA (Kornblith et al., 2019) between pretrained vision-encoder representations in Qwen2.5-VL, PaliGemma 2, and LLaVA. We sample 10 images per VisRO-Core6 task and 64 patch-token positions per image, giving 3,840 observations per backbone. Token positions are matched across depths, feature columns are centered.

Depth 0 denotes the input to the first vision block, and depth k the representation after k blocks. For Qwen, we extract the window-permuted patch embeddings and all 32 block outputs before the visual merger. For PaliGemma, we extract the patch embeddings with positional embeddings and all 27 block outputs before the connector. For LLaVA, we exclude the class token and extract representations after CLIP's pre-layer normalization and through the penultimate block, giving depths 0–23. These representations precede LLaVA's 2×2 pooling and projector. CKA characterizes pretrained representation similarity; loop spans are evaluated empirically in Section 5.2.

Trainable components. We train the question pooler, block-specific modulation generators, vision–language connector, and language-model LoRA adapters. The pretrained vision weights remain frozen unless stated otherwise. Variants A–C use the same parameterization. The loop body and modulation generators are shared across iterations, so increasing loop depth adds no parameters.

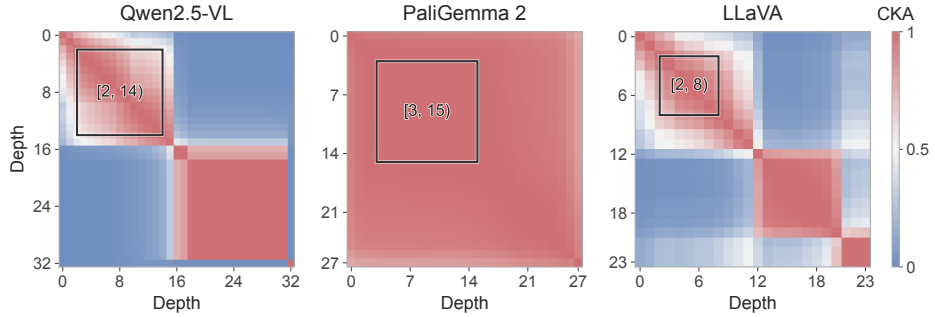


Figure 8: **Pretrained visual representation similarity.** Linear CKA across depths in Qwen2.5-VL, PaliGemma 2, and LLaVA, using the same 0–1 color scale. Boxes mark the default loop spans in Table 4; their corners correspond to each span's input and output depths. Depth k denotes the representation after k blocks. LLaVA is shown through the penultimate block used by the model. The Qwen matrix is the same as in Figure 4a.

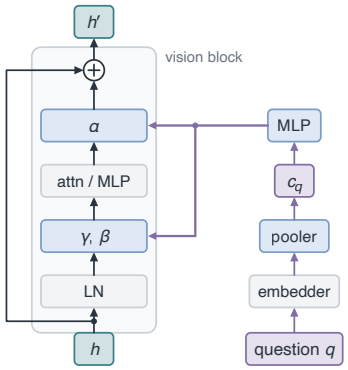


Figure 9: **Question-conditioned visual updates.** One residual branch of a frozen vision block (gray). The question is pooled into c_q and mapped to branch-specific shifts, scales, and gains. Coefficients are shared across visual tokens and iterations; the multiplicative factors are $\mathbf{1}+\gamma$ and $\mathbf{1}+\alpha$ (Equation 2). Blue modules are trainable. Only each generator's output layer is zero-initialized, preserving the pretrained computation at initialization.

Question conditioning. The pooler computes c_q from the question's input embeddings and reuses it across visual iterations. Qwen and PaliGemma omit BOS from the conditioning sequence; LLaVA prepends it. In Qwen, we additionally omit EOS and chat-role tokens, truncate the question to 128 tokens, and detach its embeddings. The embeddings are converted to FP32, RMS-normalized, and projected from width 2048 to 512. A learned query attends to this sequence through eight-head cross-attention with a residual connection, followed by a residual MLP with RMS normalization and hidden width 1024. Padding tokens are masked, and the pooler is shared across conditioned blocks. Images, answer targets, and generated drafts do not enter the question condition.

Modulation and initialization. Each block has its own generator A_ℓ, comprising layer normalization and a two-layer MLP with a SiLU activation. The output supplies a shift, scale, and residual gain for each of the attention and MLP branches, as in Equation 2. The question condition c_q and the resulting modulation coefficients are shared across visual iterations, with coefficients broadcast over visual tokens. We zero-initialize only the generator's output layer, so the first visual iteration initially recovers the pretrained encoder computation. The multiplicative factors are not squashed or clipped. Figure 9 illustrates one modulated branch.

B.2 Training

Data. The six tasks used in our experiments each contribute 100k training examples, 600 ID test examples, 600 OOD test examples and 100 validation examples used for hyperparameter tuning. Training and ID tests contain 25k and 150 examples per task and difficulty level, respectively. All task images are rendered at 448×448 before model-specific preprocessing. For Count Clusters, both test splits use icon classes held out from training; ID examples retain the training difficulty range. We mix the 600k task examples with a fixed 100k-example subset of LLaVA-NeXT, using its original answer annotations (Figure 10).

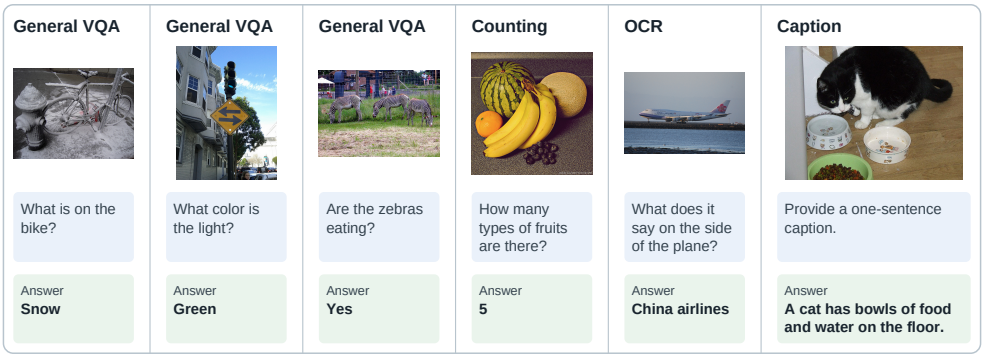


Figure 10: **Examples from the fixed LLaVA-NeXT training subset.** The examples illustrate general visual question answering, counting, optical character recognition, and captioning. Answers use the original dataset annotations (Liu et al., 2024b); prompts are shortened for readability.

Optimization. Table 4 specifies the optimization settings, with learning rates listed separately for the language-model LoRA adapters, vision–language connector, and question-conditioning modules. The latter include the question pooler and modulation generators. Within each backbone, SFT, single-pass AdaLN, and AdaLoop use the same training mixture and optimizer-update budget. Corresponding adapter initializations and training-example order are paired by seed.

Depth supervision. At each supervised depth, we teacher-force the same final answer after the variant-specific prefix $\Pi_t^{(v)}$. Only target-answer tokens contribute to the loss; question, visual, separator, and draft tokens are excluded. We use normalized depth weights $w_t \propto r^{T_{\text{train}}-t}$, where $0 < r < 1$, so later depths receive more weight. We use static geometric weights with $r = 0.6$ for Qwen and LLaVA and $r = 0.8$ for PaliGemma. Deep supervision evaluates the weighted losses at all trained depths. In our Qwen depth-sampling experiments, we instead sample a depth with probability w_t and compute its unweighted mean answer-token loss. This gives an unbiased estimate of the same objective in Section 3.3.

Drafts and gradient flow. Variant C generates its drafts greedily from the model's predictions. Drafts enter subsequent language prefixes without being replaced by the target answer or differentiated through token selection. We perform a fresh language-model prefill between rounds and detach the question embeddings used for conditioning. In all variants, gradients from the supervised loss pass through frozen vision blocks to the conditioning modules. Appendix C.2 accounts for the additional draft computation.

B.3 Inference

Inputs and decoding. On VisRO, we supply the task question through the language interfaces in Section 3.2. The visual question conditioner receives only the original question text, excluding evaluation-only answer-format instructions. We use backbone-specific causal/prefix attention conventions with the masking required by each readout variant. Final answers are decoded greedily until EOS or a maximum of 64 newly generated tokens for Qwen, 128 for PaliGemma and 32 for LLaVA.

Answer-stability stopping. Table 1 uses $p = 1$ and maximum depth $T_{\text{test}} = 3$. We obtain predictions from separate fixed-depth evaluations at depths one through three, then apply the stopping rule offline. After lowercasing answers and normalizing whitespace, we select the depth-two answer if it agrees with the depth-one answer, and otherwise select the depth-three answer. This rule uses no target answers.

Evaluation depth. Figure 5b evaluates fixed depths one through six without stopping, using models trained with $T_{\text{train}} = 3$. The mean depth $\bar{T}$ in Table 1 is the mean stopping depth selected by the rule. Appendix C.2 estimates the cost of executing this rule online, including intermediate answer generation.

Table 5: **Fixed-depth evaluation on VisRO and general VQA.** Scores are accuracies (%); VisRO scores are macro averages across six tasks. Params denotes trainable parameters (M). *General VQA evaluation uses fixed $T = 1$.

Method	Params	VisRO overall			General VQA*	
		T	**ID**	**OOD**	MMStar	MMBench
Qwen2.5-VL-3B						
SFT	66	1	49.9	23.9	51.7	77.1
Single-pass AdaLN	82	1	60.0	30.2	51.1	75.4
AdaLoop-A	82	3	63.7	32.4	51.1	75.5
PaliGemma2-3B						
SFT	24	1	62.6	31.5	46.1	72.0
Single-pass AdaLN	39	1	72.4	40.0	44.7	68.6
AdaLoop-A	39	3	74.5	44.0	44.3	71.2
LLaVA-1.5-7B						
SFT	341	1	50.4	24.8	33.7	59.1
Single-pass AdaLN	350	1	57.7	29.6	34.8	60.8
AdaLoop-A	350	3	62.1	31.1	33.7	60.1

Table 6: **Operation counts for online inference.** All variants execute the visual loop t times at fixed depth and τ times under answer-stability stopping. A readout applies the coda and connector to a visual state. A generation includes language-model prefill and answer decoding. Adaptive stopping checks the answer at every executed depth.

Variant	Fixed depth t		Adaptive depth τ	
	Readouts	Generations	Readouts	Generations
A	1	1	τ	τ
B	t	1	τ	τ
C	t	t	τ	τ

B.4 Baseline Implementations

Single-pass controls. Zero-shot evaluation uses the pretrained VLM. SFT trains the vision–language connector and language-model LoRA adapters while keeping the vision encoder frozen. Single-pass AdaLN adds AdaLoop's question pooler and modulation generators and applies the same visual span once. Its trainable parameter count matches AdaLoop's within each backbone. The question-independent control in Table 3b supplies the same null question for all training and test examples while retaining the modulation modules.

LIVR. Our Qwen implementation of LIVR (Li et al., 2026b) matches SFT in initialization, the 700k-example training mixture, sampling seed, global batch size, rank-64 language LoRA, trainable vision merger, and total 40k-update budget. Following the paper's best ablation, we use $K = 16$ learned latent tokens and train for 16k updates with visual-bottleneck attention, followed by 24k updates with standard attention. Training uses only final-answer supervision.

SFT+GRPO. We implement GRPO (Shao et al., 2024b) using EasyR1 (Zheng et al., 2025), initialized from the converged Qwen SFT checkpoint and rewarded by exact final-answer accuracy. This warm start follows preliminary runs from the pretrained checkpoint, which produced almost no correct rollouts and consequently little within-group reward variation. The GRPO stage adds training beyond the SFT budget.

C Additional Experiments and Analyses

We supplement the main experiments with fixed-depth evaluation and computational cost.

Method	Depth	TFLOPs/example	Relative FLOPs
Single-pass AdaLN	1	3.133	1.00×
AdaLoop (fixed)	3	4.170	1.33×
AdaLoop (adaptive)	≤ 3	6.777	2.16×

(a) **Inference on VisRO-ID.** Costs average 3,600 examples using text-derived generation lengths. Adaptive costs assume online stopping with $p = 1$; ratios are relative to single-pass AdaLN.

Method	Updates	TFLOPs/example	Total EFLOPs	Relative FLOPs
Single-pass AdaLN	40k	6.985	17.88	1.00×
Single-pass AdaLN	50k	6.985	22.35	1.25×
AdaLoop (sampled)	40k	9.229	23.63	1.32×
Single-pass AdaLN	120k	6.985	53.64	3.00×
AdaLoop (deep sup.)	40k	19.981	51.15	2.86×

(b) **Training on the 700k-example mixture.** Costs include forward and backward computation with global batch size 64. The sampled estimate assumes per-example state masking. Total-cost ratios are relative to single-pass AdaLN at 40k updates.

Table 7: **Estimated computational cost on Qwen2.5-VL.** AdaLoop uses variant A with loop span $[2, 14)$. Costs exclude padding and activation-checkpoint recomputation; TFLOPs and EFLOPs denote 10^{12} and 10^{18} operations.

C.1 Fixed-Depth Evaluation

Table 5 reports fixed-depth results for the same trained checkpoints evaluated in Table 1. AdaLoop uses $T = 3$ on VisRO, while all general VQA evaluations use $T = 1$. At fixed depth, AdaLoop improves over single-pass AdaLN on both VisRO-ID and VisRO-OOD across all three backbones. Changes on general VQA are mixed across models and benchmarks.

C.2 Computational Cost

We describe the FLOP accounting for the Qwen2.5-VL Variant-A comparisons in Section 5.4.

Counting convention. We count one multiply–add as two FLOPs and include matrix multiplications in the vision encoder, conditioning modules, connector, and language model. Each answer uses a KV cache, with a fresh prefill at each depth. We exclude padding, elementwise operations, normalization, softmax, activation-checkpoint recomputation, optimizer updates, data processing, and communication. These are arithmetic estimates, not latency measurements.

Inference cost. Table 6 counts operations for fixed-depth and online adaptive inference. Adaptive costs include one visual trajectory and answer generation at each depth through the selected stopping depth. They exclude repeated fixed-depth passes used for offline evaluation (Appendix B.3). Table 7a averages per-example estimates over 3,600 VisRO-ID examples, accounting for each example's context and generation lengths. We estimate generated lengths by re-tokenizing decoded predictions and adding EOS, capped at 64 tokens; these may differ from the original token counts. Online stopping uses $p = 1$ and a depth limit of three, with mean selected depth 2.19.

Training cost. Deep supervision shares one visual computation graph across depths, while each supervised depth incurs its own readout and language-model pass. The cost estimate includes backward computation through frozen vision blocks wherever it is needed to train the conditioning modules. For matrix multiplications, forward-only computation is counted once; forward plus an input gradient is counted twice; and forward plus gradients for both operands is counted three times. Thus, frozen linear layers that propagate input gradients cost twice their forward computation, while trainable linear layers and attention matrix products requiring gradients for both operands cost three times. LoRA computation is included during training and merged for inference. Intermediate language-model output heads are applied only at supervised token positions; the final head is applied to the full sequence, following the implementation.

Method	Repeated component	Initialization / adaptation
Vision models		
SR-ViT (Lee & Park, 2026)	ViT blocks	From scratch on ImageNet
LoopViT (Shu et al., 2026)	Hybrid ViT blocks	Visual ARC pretraining; test-time training
Raptor (Jacobs et al., 2026)	Surrogate ViT blocks	Fit pretrained ViT trajectories
Language models		
Huginn (Geiping et al., 2025)	Middle LM blocks	From scratch
Ouro (Zhu et al., 2025)	Full LM blocks	From scratch
LoopUS (Park et al., 2026b)	Middle LM blocks	Continued LM pretraining
Multimodal models		
RecursiveVLM (Xu et al., 2026b)	Full LM blocks	Continued VLM pretraining + SFT
HIVE (Zhang et al., 2026a)	Huginn recurrent core	Pretrained vision + LM; multimodal training
CVRR (Park et al., 2026a)	Single LM block	Pretrained VLM; recurrent LoRA
AdaLoop	Frozen vision blocks	Train conditioning, connector, LM LoRA

(a) **Looped architectures.** Parameters are shared across repeated applications of the listed component.

Method	Mechanism	Conditioning / feedback
Mema (Liu et al., 2026b)	Cross-layer memory + modulation	Input question
VPT (Yu et al., 2025)	Re-encoding; conditioned projection	LM control-token state
Galatolo et al. (2026)	Feedback to encoder inputs	LM image-token states
DVR (Zhang et al., 2026b)	Re-encoding	Generated sub-questions
AdaLoop	Recurrent visual updates	Input question

(b) **Language-guided visual processing.** Visual computation can be conditioned directly on the question or on intermediate language-model states or outputs.

Table 8: **Looped architectures and language-guided visual processing.** The two panels compare complementary architectural choices across different tasks and training settings. LM denotes language model.

For depth-sampled supervision, we estimate the cost of the example-masked implementation. It masks state updates after each example's sampled depth but still executes all iterations up to the training limit. The estimate includes this visual computation and one supervised readout at the sampled depth.

We estimate total training FLOPs by averaging per-example costs over the training mixture and multiplying by the global batch size and number of updates (Table 7b). The sampled and deeply supervised runs use 5.7% more and 4.6% less compute than their respective AdaLN controls.

D Additional Related Work

Reusing pretrained visual computation. Table 8a compares recurrent architectures, and Table 8b compares how language guides visual processing. Raptor (Jacobs et al., 2026) learns recurrent surrogate blocks to approximate pretrained ViT activation trajectories. Mema (Liu et al., 2026b) maintains question-conditioned memory across encoder layers. AdaLoop repeatedly applies the original frozen vision blocks to an evolving visual state. Question-conditioned modulation guides these updates, with generator parameters shared across iterations.

Relation to procedural training resources. VisRO shares with VBVR and VBVR-Pro an emphasis on procedural data generation and programmatic evaluation (Wang et al., 2026a; Xu et al., 2026a). These works evaluate generated visual solutions. Our experiments use image–question–answer triples and final-answer supervision, without prescribed visual solutions or intermediate trajectories.

Table 9: **VisRO task coverage.** The 24 tasks are grouped by the visual operations they require. Questions and answer formats correspond to representative examples in Figures 3, 11, and 12; difficulty extensions can use different formats.

Task	Representative question	Answer form
Discrimination and grouping		
Count clusters	How many objects overlap another object?	Integer
Points on lines	How many points lie on the highlighted lines?	Integer
Find the same	Which items are identical?	Correspondences
Find the different	Which item differs from the others?	Position
Find the shadow	Which silhouettes match the objects?	Correspondences
Overlapping glyphs	Read symbols in the indicated color order.	Symbol sequence
Overlap matching	Which outline occurs in the superposition?	Choice
Paths and connectivity		
Flow Free	Count cells per pipe, including endpoints.	Integer sequence
Metro map	How many stations are on the specified line?	Integer
Maze	Are the two marked locations connected?	Boolean
Path tracing	Match the marked starts to endpoint labels.	Label sequence
Planar transformations and composition		
Jigsaw completion	Which piece completes the pattern?	Choice
Reconstruction	Which set of pieces can form the target?	Choice
Tangram reconstruction	Which pieces form the target silhouette?	Choice
Mirroring patterns	What pattern results from the given reflections?	Choice
Rotation patterns	Which patterns result from the given rotations?	Index sequence
Logic matrix	Which entry completes the matrix rule?	Choice
Spatial transformations and structure		
3D block counting	What minimum cube count is consistent with the view?	Integer
Block changes	How many cubes were added and removed?	Ordered counts
Cube unfolding	Which cube views can be folded from this net?	Index set
Paper folding	What pattern results after folding, punching, and unfolding?	Choice
3D views	Which projection matches the indicated viewpoint?	Choice
View reconstruction	Which structure is consistent with the given views?	Choice
Viewpoint rotation	Which candidate is a rotation of the reference object?	Choice

E VisRO: Tasks, Generation, and Evaluation

VisRO is a procedural dataset and generation pipeline with 24 visual reasoning tasks spanning counting, discrimination, connectivity, geometric transformations, and spatial reasoning. This appendix describes the tasks, their difficulty controls, and the data generation and evaluation protocol.

E.1 Task Coverage and Definitions

Table 9 groups the 24 tasks by the visual operations they require: discrimination and grouping, paths and connectivity, planar transformations and composition, and spatial transformations and structure. Figures 3, 11, and 12 together illustrate all tasks. Questions specify both the requested operation and the answer format. The table and figures illustrate representative formats. Some difficulty extensions use different outputs, including per-line counts, queried-cell values, and constructive tangram assemblies.

E.2 Difficulty Levels and OOD Evaluation

Difficulty levels. Training and in-distribution (ID) evaluation use four task-specific difficulty levels: easy, medium, hard, and xhard. ID evaluation uses independently generated test examples from these four levels. Difficulty is controlled through generation parameters such as object count, overlap, path length, crossings, and distractor similarity. Table 10 compares xhard, the highest ID level,

Table 10: **Example ID and OOD settings in VisRO.** We compare xhard, the highest ID difficulty, with the held-out OOD settings. A trailing + denotes a lower bound. In cube unfolding, OOD items distinguish same-color faces by glyph identity and orientation.

Task	ID (xhard)	OOD
Find the same	20 icons; 6 matching pairs; 6 near matches	25 icons; 8 matching pairs; 8 near matches
Find the different	9×9 grid; local alteration	10×10 grid; smaller alteration cap
Cube unfolding	10 views; orientation distractors	12 views; glyph-orientation cues
3D block counting	4×5 base; max height 5; min count 30+	5×5 base; max height 6; min count 45+
Block changes	8–10 changes; 5+ columns	12–16 changes; 8+ columns
Viewpoint rotation	4×5 base; 20–28 cubes	5×5 base; 30–42 cubes

with the OOD settings for representative tasks. Tier names describe task-specific settings and do not imply equal difficulty across tasks.

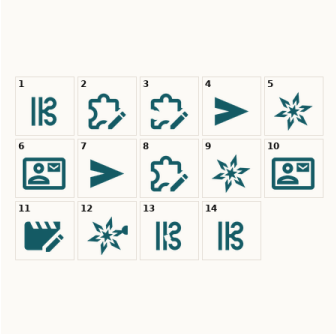
Difficulty controls also account for the rendered problem. For example, near-matching icons increase comparison demands, while cube-net distractors can differ in face orientation. These changes must preserve the visual cues needed to determine the answer. Rendering resolutions are task-specific and specified by the generator configurations.

The OOD split. Out-of-distribution (OOD) examples introduce task-specific increases in scene complexity or shifts in generation rules and visual appearance. For example, 3D block counting increases the base dimensions, maximum height, and minimum cube count (Table 10). Flow Free holds out a path-construction program and rendering or material combinations; Metro Map uses four held-out rendering styles. Both tasks retain their ID question families in the OOD split. Figure 3 shows representative ID and OOD examples.

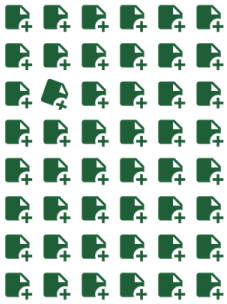
E.3 Data Generation and Evaluation

Data generation and validation. Each generator samples and renders a structured scene, then constructs a question and computes its answer from the scene. Task-specific checks assess whether the image contains the information needed to answer the question and whether the answer agrees with the scene. Scene structure and spatial annotations support data generation and validation. AdaLoop training uses only images, questions, and final answers.

Scoring. Scoring follows each task's answer format, without an LLM judge. Labels, counts, and sequences use exact match after task-specific parsing and normalization. Ordered outputs must match the complete reference sequence, while pair sets are compared independently of their listing order. Constructive tangram answers are checked for allowed rotations, complete target coverage, and absence of overlap. Any valid assembly is accepted. We report per-task accuracy and macro averages across evaluated tasks, separately for ID and OOD.



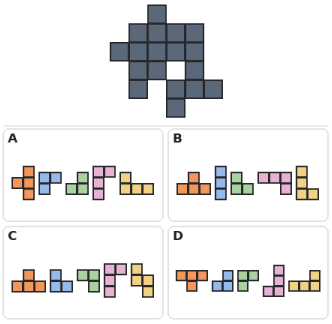
Find the same
Which icons are identical?



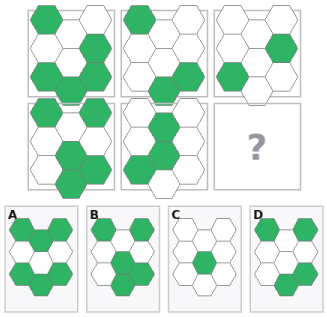
Find the different
Which icon is different?



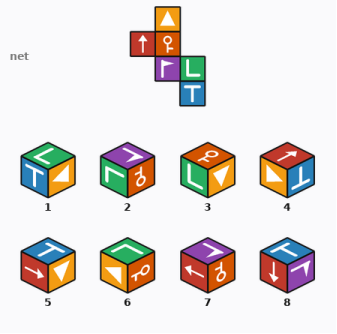
Find the shadow
Match objects to silhouettes.



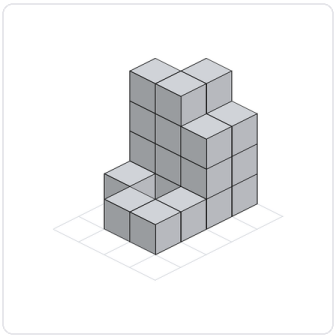
Reconstruction
Which pieces form the target?



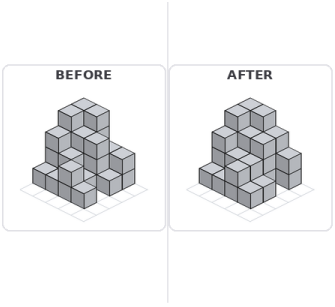
Logic matrix
Complete the missing entry.



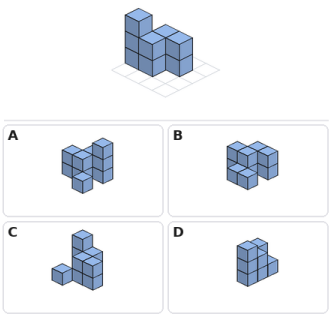
Cube unfolding
Which cubes match the net?



3D block counting
Find the minimum cube count.

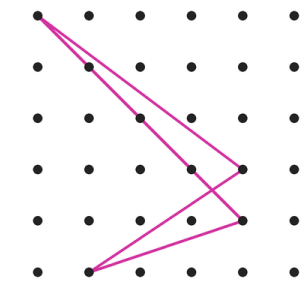


Block changes
Count added and removed cubes.

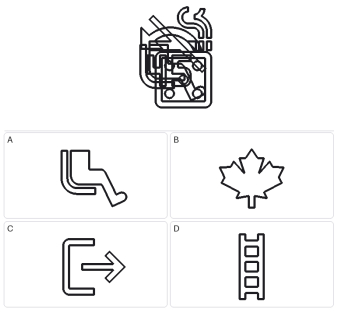


Viewpoint rotation
Find the same object after rotation.

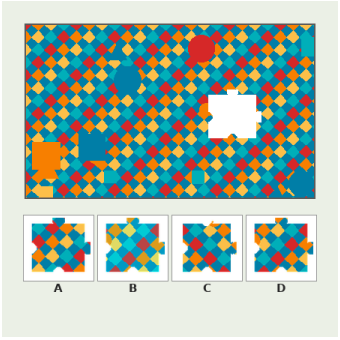
Figure 11: **VisRO task examples (1/2).** The examples illustrate matching and discrimination (top), composition and transformations (middle), and spatial structure (bottom). All examples use the task-specific *hard* setting from a fixed generator revision; the text below each image summarizes its question.



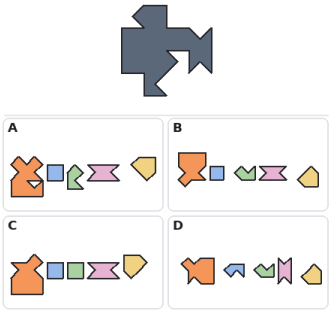
Points on lines
Count points on the colored lines.



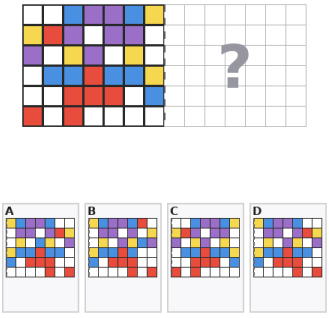
Overlap matching
Which outline appears in the stack?



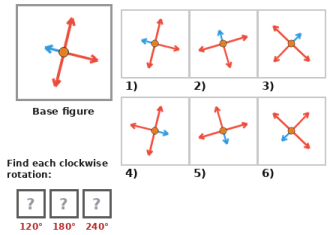
Jigsaw completion
Which piece completes the pattern?



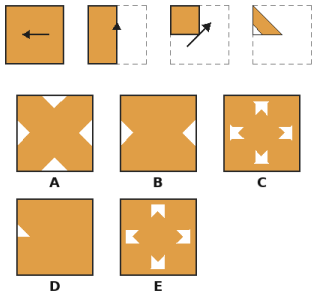
Tangram reconstruction
Which pieces form the silhouette?



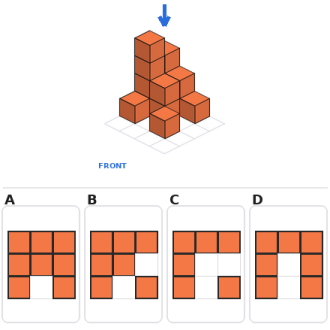
Mirroring patterns
Complete the reflected pattern.



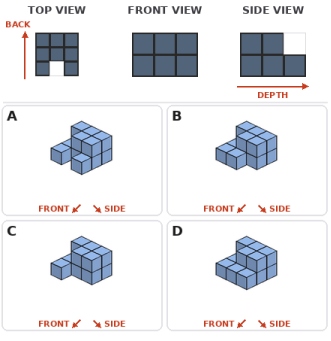
Rotation patterns
Match the requested rotations.



Paper folding
Unfold the cut paper.



3D views
Find the indicated projection.



View reconstruction
Which structure matches the views?

Figure 12: **VisRO task examples (2/2).** The examples illustrate point counting, overlap matching, and jigsaw completion (top), planar composition and transformations (middle), and folding and three-dimensional views (bottom). As in Figure 11, all examples use the task-specific *hard* setting, and the text summarizes each question.