RETROFITTING RECURRENT VISION–LANGUAGE MODELS FROM PRETRAINED BACKBONES

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent work shows that depth recurrence can be retrofitted into pretrained language models. In this work, we adapt this paradigm for visual understanding, using depth recurrence to reduce the parameter footprint of a pretrained vision-language model (VLM). To this end, we prune layers from the language backbone of the VLM and loop over a subset of the remaining layers. At each step of the loop, a linear adapter merges the block's output with the token representations from the preceding layers and projects the fused representation back into the block. As the pruning impacts the original model performance, we propose a three-stage recovery training: first, we leverage a language-only pretraining to recover the performance of the language backbone, followed by an alignment of the vision-language projector, and a full fine-tuning of the model via visual instruction tuning. In the first and the last stage, we distill the outputs of the pretrained non-recurrent model into the recurrent model, which further improves the recovery. We evaluate the proposed approach on the TinyLLaVA-1.4B backbone without introducing any new vision-language data during retrofitting. Our experiments demonstrate that the retrofitted recurrent model outperforms the unpruned baseline trained with the same recovery data while storing 23.3% fewer language parameters. We further show that the proposed retrofitting recurrence outperforms the setup of converting a recurrent LM into a VLM on the same data. Additionally, our ablations show how design choices, such as the adapter between recurrences and a larger recurrent block, lead to the improved performance of our approach. Finally, we find that our proposed model gains on reasoning-oriented benchmarks, whereas its performance on basic visual understanding benchmarks remains on par with the baseline.

1 INTRODUCTION

Existing approaches for test-time compute scaling in vision-language models comprise chain-of-thought traces, latent reasoning tokens, searching over candidate answers, and test-time reinforcement learning (Zhang et al., 2024b; Qin et al., 2025; Xu et al., 2025; Singh et al., 2025; Wan et al., 2026). These methods usually come with higher inference costs as each generated token requires a full forward pass and expands the context window. To address this limitation, *depth recurrence* (Geiping et al., 2025; McLeish et al., 2025) has been explored as an alternative for test-time scaling in language models. Depth recurrence allocates test-time compute to iterating a block of layers rather than generating tokens, thereby maintaining smaller context sizes and a smaller memory overhead independent of reasoning length.

But depth recurrence has also gained traction in visual understanding. Here, recurrent VLM architectures are built in two ways: first, either by starting from a recurrent language model and post-hoc adding a vision backbone, or, second, by looping an existing VLM without removing any of its layers. The first approach is i.a. adopted by HIVE (Zhang et al., 2026), which extends a recurrent language model pretrained from scratch by a vision backbone. Examples for the second approach are, RecursiveVLM (Xu et al., 2026) and ReLoop-UME (Wang et al., 2026), which loop layers in pretrained VLMs without pruning, with a near-constant number of parameters.

In contrast to these two regimes, we pursue a third option of retrofitting a pretrained VLM with recurrence, following a prune-then-loop strategy. Namely, we remove some intermediate blocks of the language backbone of the VLM and loop over a subset of the remaining layers. This strategy

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.

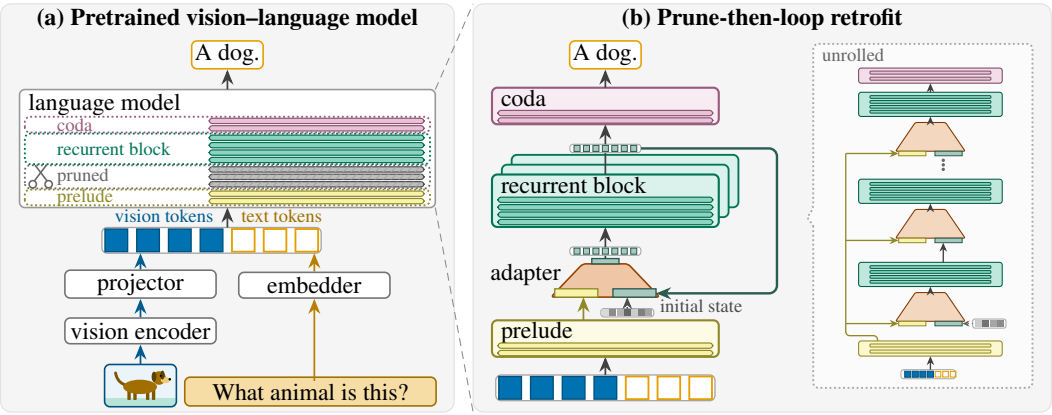


Figure 1: Prune-then-loop retrofit. Left: a pretrained VLM embeds the image as vision tokens, the prompt as text tokens and passes them through its language backbone. The layers marked in grey are pruned. Right: Our retrofitting mechanism reuses the remaining layers. The prelude is executed once, and an adapter merges the prelude's output with the hidden state of the recurrent block initialized with noise. The adapter and recurrent block are applied repeatedly, with the coda used to decode to logits.

is supported by two complementary research findings: first, one set of recent works shows that a fraction of a VLM's layers can be removed at a limited cost in accuracy (Huang et al., 2026; Madinei et al., 2025; Ma et al., 2025), and, second, another set of works shows that the layers of a pretrained model can be skipped or repeated at inference (Li et al., 2026; Chen et al., 2026). We thus hypothesize that iteratively executing the subset of layers of a pruned VLM can recover the capacity lost due to pruning. We test this hypothesis by *retrofitting*, i.e., by converting a pretrained non-recurrent VLM into a recurrent one while reusing its existing weights rather than pretraining a recurrent model from scratch. We refer to the resulting model as retrofitted recurrent VLM (RR-VLM). We prune a block of intermediate layers from the language backbone of a pretrained VLM and designate a contiguous subset of the remaining layers as a recurrent block (McLeish et al., 2025). To feed the block's output back into its input for successive iterations, we concatenate the output with the tokens from the layers preceding the recurrent block and train a linear layer to compress this concatenated representation to its original dimensionality (McLeish et al., 2025). An overview of the process is given in Figure 1. This surgery, however, degrades the pretrained language backbone, and although text-only training repairs it, the repair leaves the language model misaligned with the original projector. We resolve this with a three-stage training curriculum: first, we apply a text-only training to repair the backbone, second, we apply a projector-only training to restore the image-text alignment, and third, we fine-tune the entire model on visual instruction data. In the text-only and the instruction-tuning stage, we add a knowledge distillation (KD) term, with the pretrained model as teacher, to avoid the language backbone from drifting too far from the original representation that the projector was aligned with.

We evaluate the proposed method on the TinyLLaVA-1.4B backbone (Zhou et al., 2024), training it on 60B tokens of text as well as 493k image–caption pairs and 660k instruction tuning samples from the base model's training data, without introducing any new vision-language data. We compare the resulting model against the pretrained TinyLLaVA-1.4B model and four baselines trained on the same data: (1) the unpruned model subjected to the identical recovery training, (2) the unpruned model without the text-only stage, (3) the pruned model without recurrence, and (4) a recurrent language model that has been extended to a VLM. Our experiments show that our proposed retrofitted recurrent model outperforms the unpruned baseline while having 23.3% fewer language-model parameters and show that this gain is due to the recurrence rather than the recovery training. The gain concentrates on reasoning-oriented benchmarks, whereas performance on basic visual understanding benchmarks remains on par with the baseline. Compared to a recurrent language model extended to a VLM, our retrofitted model achieves a higher average accuracy. We further ablate each component of the retrofit: the training stages, the layer allocation, the knowledge distillation term, KV-cache sharing at inference and the adapter. We find that the text-only stage S_0 is necessary for the recovery, as removing it costs 3.68 pp, primarily on the reasoning-oriented benchmarks. Specifically, we evaluate our linear adapter against three variants: a gated adapter, the Parcae adapter (Prairie et al., 2026), which multiplies the recurrent state with a learned per-channel factor within

$(0, 1)$ so that the hidden state remains bounded across recurrences, and no adapter at all. Our ablations show that, among the adapter variants, the linear adapter yields the best performance, whereas the gated adapter and the Parcae adapter score below the variant without an adapter. Furthermore, we show that at equal compute, a larger recurrent block results in better performance. We also show that distillation from the pretrained model improves recovery training and increases the performance of the retrofitted recurrent model. Finally, we find that a model trained at a fixed depth only works at the depth it was trained at, whereas a model trained with sampled recurrence works at other depths, but its performance does not improve beyond the trained depth. We also find that sharing the KV cache across recurrences during inference costs accuracy, which contrasts with the results reported for a model pretrained with recurrence (Geiping et al., 2025).

We summarize our contributions as follows: (1) We introduce the first prune-then-loop retrofit of a pretrained VLM evaluated on VQA, supported by a three-stage curriculum that repairs the pruned language backbone and restores projector alignment using the base model's vision-language training data. (2) We demonstrate that retrofitting recurrence directly into a pretrained VLM outperforms the alternative of adding a vision encoder to a pretrained recurrent language model. (3) We perform an extensive evaluation of the retrofitted recurrent VLM and quantify the impact of each design decision.

2 Related Work

Depth recurrence and architectural retrofitting. Depth recurrence reuses a block of layers multiple times, increasing the effective depth of the model without adding any additional parameters (Dehghani et al., 2019). Previous attempts at creating recurrent LMs include Huginn (Geiping et al., 2025) and Ouro (Zhu et al., 2025). Huginn splits the model into three components: a *prelude* that embeds the input into a latent state, a *recurrent block* that iterates on that state, and a *coda* that decodes the final state into output logits. Ouro iterates the entire set of layers, and Loopie (Gao et al., 2026) applies each layer recurrently instead of performing block-level looping. In all three cases, the model is trained from scratch without any pre-initialization by existing models. Two findings inform our design: a looped block benefits from a stable learned connector between successive passes (Prairie et al., 2026), and an iterated layer contributes only a fraction of the capacity of a stored one (Schwethelm et al., 2026), thus bounding what recurrence can recover from pruning. As an alternative to full-corpus pretraining, recurrence can instead be achieved by retrofitting a pretrained model. Within this regime, there exist two lines of work: (1) looping a middle block of the stack without pruning, for latent reasoning (Koishekenov et al., 2025), and (2) looping after reducing the depth, either by keeping a block of the surviving layers (Bae et al., 2025; McLeish et al., 2025) or by merging neighbouring layers into a few blocks that are then looped (Kwon & Park, 2026). We adopt the latter approach, retrofitting recurrence after structural pruning. Specifically, we take inspiration from McLeish et al. (2025), who drop layers 4 to 9 of TinyLlama-1.1B to obtain a $4/8/4$ layer allocation looped through a linear adapter.

Recurrence in vision-language models. Current works either loop layers of a pretrained VLM without pruning, with a near-constant number of parameters (Xu et al., 2026; Wang et al., 2026), or build one on a natively recurrent language backbone (Zhang et al., 2026). HIVE (Zhang et al., 2026) builds a VLM on the pretrained recurrent language model Huginn-3.5B (Geiping et al., 2025) by attaching an InternViT-300M vision encoder and training the combined model on 6.5M training samples in three stages. As such HIVE is based on a significantly larger model and trained on five times more training samples than the proposed model, preventing a direct comparison to those checkpoints. We therefore take inspiration from the pipeline in HIVE to replicate this setting as a baseline to compare to rather than comparing against their model. Beyond standard vision-language tasks, depth recurrence has been explored for robotic action policies, either in the action head of an intact VLM (Tur et al., 2026) or by pruning a VLM and looping a small block (Shen et al., 2026). However, these methods operate in a fundamentally different domain, and the pruned variant of Shen et al. (2026) lacks a prune-only baseline. Our setting departs from those existing approaches by following a prune-then-loop pattern, removing layers before the loop is installed. To the best of our knowledge, this work presents the first prune-then-loop retrofit of a pretrained VLM evaluated on VQA.

3 METHOD: PRUNE-THEN-LOOP RETROFITTING

We study the problem of converting a pretrained non-recurrent vision-language model into a depth-recurrent architecture. In this section, we formalize the prune-then-loop retrofit, explaining the proposed model surgery, the training curriculum used to recover performance, and the adapter mechanism.

3.1 RETROFITTING EXISTING VLMS

A pretrained vision-language model computes next-token logits from an image I and a text sequence x as

$$F(I, x) \;=\; L(u), \qquad L = \ell_m \circ \cdots \circ \ell_1, \qquad u = [\,\pi(V(I));\; E(x)\,], \tag{1}$$

where V denotes the vision encoder, π the projector, E the text token embedder, and L the language model composed of layers operating on representations of dimension H. Our prune-then-loop mechanism modifies only the layers $\ell_1, \ldots, \ell_m$ of the language model L. We partition the m layers into four consecutive groups of sizes $p + d + b + c = m$ and delete the d layers. This yields a *prelude* block $P = \ell_p \circ \cdots \circ \ell_1$, a *recurrent* block $B = \ell_{p+d+b} \circ \cdots \circ \ell_{p+d+1}$, and a *coda* block $C = \ell_m \circ \cdots \circ \ell_{m-c+1}$ (Figure 1).

The retrofitted model applies the recurrent block r times. The recurrent block is preceded by a small adapter A that merges the block's current output with the output of the prelude. Expressing this pair as $B \circ A$ and repeating it r times yields:

$$F^{r}_{p/b/c} \;\coloneqq\; C \circ (B \circ A)^r \circ P. \tag{2}$$

Namely, we first apply the prelude P, then run the combination of adapter and recurrent block $B \circ A$ for r times, and apply the coda C once. We refer to $F^{r}_{p/b/c}$ as the retrofitted recurrent VLM (RR-VLM). The prelude P computes a context h from the token input u as $h = P(u)$. The adapter A then concatenates the tokens of the recurrent block output s with the tokens of prelude context h, resulting in the same number of tokens with double dimension, and then compresses the token-wise concatenated representation back to the original dimension H through a learned linear layer $W \in \mathbb{R}^{H \times 2H}$ (McLeish et al., 2025). In case of the first loop, when no recurrent output is computed yet, the tokens s_0 are initialized from a normal distribution $\mathcal{N}$ (Geiping et al., 2025) and k is defined as the recurrence step:

$$\begin{aligned} A(s, h) &= W[s; h], && && \text{linear adapter } A_{\text{lin}} \\ s_k &= B\big(A(s_{k-1}, h)\big), && s_0 \sim \mathcal{N}(0, \sigma^2 I), && \text{recurrent block, } k = 1, \ldots, r \\ y_k &= C(s_k), && && \text{readout after } k \text{ recurrences} \end{aligned} \tag{3}$$

The coda C maps the state after k recurrences to output logits y_k, and the output of the model is y_r. Note that every input token traverses $m^r = p + rb + c$ layers, but the model stores only $p + b + c$ distinct layers. Restricting this traversal to r=1 defines the single-pass model F^1.

Number of recurrences. The number of recurrences r can be chosen flexibly during training and inference. During training, it can be defined in two ways: (1) as a constant, defining the model F^r, or (2) as a per-step sample from a log-normal Poisson distribution (Geiping et al., 2025) with mean $\bar{r}$, defining the model $F^{\bar{r}}$. At inference, r is set to a fixed value that can be independent of the training configuration.

KV cache. Each pass through the recurrent block computes its own keys and values, so the KV cache holds m^r entries per token compared to the m entries for the original model. We evaluate two ways of sharing the cache across passes. The first, from Geiping et al. (2025), keeps n copies of the block's cache per token and lets pass i use copy $i \bmod n$. With n=1, every pass reads the entries that the last pass wrote, and the cache shrinks to $p + nb + c$ entries per token. This variant changes only the inference procedure and requires no training. The second lets every pass reuse the keys, or the keys and values, that the first pass computed, so that the cache shrinks to $p + b + c$ entries per token. This variant changes the forward pass itself. The model is therefore trained with the sharing active and evaluated with the same forward pass.

3.2 Training Strategy

The retrofitted model of Sec. 3.1 requires training for two reasons. First, the pruning removes the layers between the prelude and the recurrent block, leading to a misalignment between the representations produced by the prelude and those expected by the recurrent block. Additionally, the remaining layers have to be adapted to the modified architecture. Second, the recurrence introduces an adapter A that fuses the output of the prelude with the output of the recurrent block using a newly initialized linear layer that needs to be trained as well.

Three-stage Training. We propose a three-stage curriculum that restores the VLM and trains the adapter: First, in stage S_0 we focus on repairing by training only the pruned language backbone on text-only data, freezing all other components. Second, as the retrained text backbone can now slightly deviate from the original one, we retrain the vision-language projector in stage S_1 to restore image-text alignment, and finally optimize the entire model via end-to-end training with visual instruction data in stage S_2, interleaving image-text and text-only training data.

Knowledge Distillation. Retraining the modified language backbone inevitably shifts its internal representations away from those of the pretrained model. In a VLM this shift is problematic, since the vision-language projector has been aligned to the representations of the original backbone. We therefore seek to adapt the language backbone to the new architecture while preserving its original representations as far as possible. To this end, we incorporate a distillation term at stage S_0 training, using the original, unpruned baseline model F as the teacher to limit the drift of the pruned, recurrent model F^r and to allow an easier realignment in stage S_1:

$$\mathcal{L} \;=\; (1-\lambda)\,\mathcal{L}_{\mathrm{CE}} \;+\; \lambda\,\tau^2\;\mathrm{KL}\big(\,\mathrm{softmax}(F/\tau)\;\big\|\;\mathrm{softmax}(F^r/\tau)\big). \tag{4}$$

The distillation objective evaluates a single pass of the teacher against the iterative execution of the student. Note that in stage S_0, this is done on text-only training data. We found that such knowledge distillation can further improve performance during stage S_2 training, as this combines the vision and language input signal from the teacher.

4 Implementation Details

Base model. We apply the retrofit approach to the pretrained `bczhou/tiny-llava-v1-hf` checkpoint (Zhou et al., 2024), named TinyLLaVA-1.4B by its authors, which comprises a 22-layer TinyLlama-1.1B language model (Zhang et al., 2024a), a CLIP ViT-L/14-336 vision tower (Radford et al., 2021), and a LLaVA-style projector (Liu et al., 2023). Note that McLeish et al. (2025) retrofit the same language model in the text-only setting into a recurrent language model. This allows us to use this as a base model for a directly comparable baseline of a recurrent language model that is extended by a vision backbone.

Model Surgery. We again follow McLeish et al. (2025) and delete six contiguous layers 4-9 (d=6) from the pretrained TinyLLaVA-1.4B checkpoint, leaving a 16-layer language model with the layer allocation $p/b/c = 4/8/4$. The final model uses a fixed recurrence of r=6, resulting in each token traversing an equivalent depth of $m^r = 4 + 6 \times 8 + 4 = 56$ layers. After surgery, the language model including the adapter contains 844.4M parameters compared to the 1,100.3M of the language backbone of the base model (-23.3%). The adapter is a single linear layer with 8.4M parameters. Including the vision encoder and the projector the model has 1,154.2M parameters compared to 1,410.1M of the base model (-18.1%).

Training. The training follows the three-stage curriculum of Sec. 3.2. All three stages use Muon (Jordan et al., 2024) on matrix parameters and AdamW (Loshchilov & Hutter, 2019) on the remainder. The text-only training phase S_0 uses a 60B-token blend of 85% SlimPajama (Soboleva et al., 2023) and 15% FineWeb-Edu (Penedo et al., 2024) for the main experiments, and on 7.5B tokens of SlimPajama for our ablations. The text-only training phase adds the KD term of Eq. 4 with λ=0.5 at τ=2. The image-text alignment phase S_1 trains on the base model's alignment set of `blip_sbu_cc` (Li et al., 2022) containing 493k image-caption pairs. It updates only the projector π and adds no KD term. The visual instruction tuning phase S_2 finetunes the entire model on the base model's instruction data of `llava_instruct` (Liu et al., 2023) which consists of 660k cleaned samples, interleaving four multimodal batches and one text batch (Appendix A.4 varies this ratio). It applies the KD term again, with a separate weight and temperature for each batch type: λ=0.25 at τ=2 on multimodal batches and λ=0.5 at τ=1 on text batches. All models are trained with seed 42.

Table 1: Comparison to baselines. The retrofit outperforms the baselines on average over nine benchmarks with 23.3% fewer stored language-model parameters. Sec. 5.1 defines the models. RR-VLM denotes our retrofitted recurrent VLM $F^{6}_{4/8/4}$ of Sec. 3 and F^{1} the pruned single-pass model.

Model	LM (B)	GQA	POPE	TVQA	SQA	MMB	RWQA	SEED	AI2D	CV2D	Avg
Baseline											
TinyLLaVA-1.4B	1.10	57.6	85.1	46.7	59.6	49.7	45.4	54.1	39.4	38.6	52.91
Baselines, no recurrence											
Baseline+(S_1, S_2)	1.10	59.9	**86.9**	45.5	57.8	55.6	47.3	59.8	41.3	43.4	55.27
Baseline+(S_0, S_1, S_2)	1.10	**60.4**	86.7	47.3	**60.2**	55.5	46.4	60.4	43.2	42.2	55.80
Baseline(pruned)+(S_0, S_1, S_2) (F^1)	0.84	58.0	86.1	37.7	50.3	48.6	47.6	58.3	38.1	47.1	52.41
Converted recurrent LM (McLeish et al., 2025)											
Recurrent LM+ViT+(S_1, S_2)	0.84	58.8	86.3	41.5	52.5	52.7	**50.6**	60.7	43.6	53.7	55.59
Pruned and looped (ours)											
RR-VLM $F^{6}_{4/8/4}$	0.84	59.2	86.7	**47.7**	59.1	**58.9**	50.5	**62.6**	**44.6**	**53.9**	**58.13**

5 Evaluation

Benchmarks. We evaluate the models on the following nine benchmarks: GQA (Hudson & Manning, 2019), POPE (Li et al., 2023b), TextVQA (TVQA) (Singh et al., 2019), and ScienceQA-IMG (SQA) (Lu et al., 2022), MMBench (MMB) (Liu et al., 2024), RealWorldQA (RWQA) (xAI, 2024), SEED-Bench (SEED) (Li et al., 2023a), AI2D (Kembhavi et al., 2016), and the 2D split of CV-Bench (CV2D) (Tong et al., 2024). We use the VLMEvalKit (Duan et al., 2024) library for all model evaluations. We define the Avg metric as the arithmetic mean of the nine VQA benchmarks. We set a fixed eval seed to make evaluation deterministic.

5.1 Comparison to Baselines

Baselines. We evaluate the proposed recurrence retrofitting approach against five baselines and report the results in Table 1. The first baseline is the pretrained TinyLLaVA-1.4B checkpoint without any modifications. The second and third baselines fine-tune this model without pruning, either with the image-text stages S_1 and S_2 only or with all three stages S_0, S_1 and S_2 of our training strategy. The fourth baseline applies all three stages to the pruned model without recurrence, which corresponds to the single-pass model F^1 of Sec. 3.1. The baselines with the text-only stage apply the distillation term of Eq. 4 in S_0, and the unpruned Baseline+(S_0, S_1, S_2) also applies it in S_2, so that it is trained with the full strategy of our model. We include those retrained versions to allow a direct comparison between the retrofitted recurrent model and the same non-recurrent backbones trained on the same data. To allow a comparison to a setup that starts from a recurrent language model and adds the vision backbone after recurrent training, we convert a pretrained recurrent TinyLlama checkpoint of McLeish et al. (2025) into a recurrent VLM by adding the pretrained image encoder from TinyLLaVA-1.4B and train the model with the same image-text data on the training stages S_1 and S_2. For our retrofitted recurrent VLM (RR-VLM), we use a fixed recurrence of $r{=}6$ and report results for an 8-layer recurrent block architecture ($F^{6}_{4/8/4}$).

We see that the training improves the baseline models by around 2.4 to 2.9 pp, while the performance of the pruned model is slightly below that of the base model. The performance of the converted recurrent VLM is in the same range as the retrained baselines. Compared to that, the proposed RR-VLM model is able to outperform all baseline models on average by at least 2.3 pp. Analyzing the performance per dataset, we see that more basic visual understanding datasets, such as GQA, POPE, TVQA and SQA, do not profit as much from the recurrent architecture, and that the RR-VLM performs on par with the baseline model. This can be attributed to the fact that those datasets usually have single-step QA type questions, which do not profit from more complex reasoning. Recurrence adds computation without adding information, so no gain is expected on such questions.

Compared to that, we see a clear trend of performance increase on more reasoning-oriented datasets such as MMBench, RealWorldQA, SEED-Bench, AI2D and CV2D. Note that on the latter four datasets, both recurrent models, the converted VLM as well as the proposed RR-VLM model, perform consistently better than the non-recurrent baselines, with RR-VLM still outperforming the post hoc converted VLM on three of the four datasets. We thus find that retrofitting a pretrained VLM results in better performance than introducing a vision encoder into a recurrent language model to create a recurrent VLM. We thus find that retrofitting a pretrained VLM can result in improved

Table 2: Training stages. We remove single stages of the training strategy of Sec. 3.2 from our best performing setting ($4/8/4$, r=6, 60B S_0 budget), each stage with its distillation term. The first row repeats the unpruned baseline of Table 1, which is trained with the full strategy. Avg is the mean of the nine columns to its left.

Model and training stages	GQA	POPE	TVQA	SQA	MMB	RWQA	SEED	AI2D	CV2D	Avg
Baseline+(S_0, S_1, S_2)	60.4	86.7	47.3	60.2	55.5	46.4	60.4	43.2	42.2	55.80
RR-VLM, S_1, S_2	59.6	86.5	45.5	54.6	52.1	44.1	61.0	40.0	46.6	54.45
RR-VLM, S_0, S_2	59.3	86.6	47.0	57.1	56.7	48.8	62.4	42.7	51.2	56.85
RR-VLM, S_0, S_1, S_2 (ours)	59.2	86.7	47.7	59.1	58.9	50.5	62.6	44.6	53.9	**58.13**

Table 3: Ablations. Each table changes one setting of the model named in its header and the best value is in bold. All tables (a), (b) and (c) use the 7.5B S_0 budget without text distillation.

(a) Layer allocation, 56 layers	
$6/4/6$, r=11	52.56
$4/6/4$, r=8	54.15
$4/8/4$, r=6	**55.75**

(b) Adapter, $4/6/4$**,** r=6	
$A_\varnothing$ (none)	54.29
A_{sc}	52.98
A_{pc}	53.76
A_{lin}	**55.02**

(c) Looping variant, $4/6/4$	
per-layer looping, r=2	52.19
per-layer looping, r=6	49.62
block looping, r=6	**55.02**

performance compared to introducing a vision encoder into a recurrent language model to create a recurrent VLM.

5.2 Ablations

Training stages. We determine the effect of the text-only stage S_0 and the vision-language alignment stage S_1 on the resulting recurrent VLM in Table 2. We find that S_0 is crucial to the recovery training, as removing it reduces the performance of the resulting model by 3.68 pp on average. Removing S_0 primarily affects the reasoning-oriented datasets, with 7.3 pp lost on CV2D, 6.8 pp on MMB and 4.6 pp on AI2D. We attribute this to the fact that without the repair of the pruned language backbone in S_0, the stages S_1 and S_2 have to repair the language backbone and align the projector at the same time. We further find that removing S_1 costs 1.28 pp, again primarily on the reasoning-oriented datasets. We attribute this to the fact that S_1 performs the vision-language alignment, and without it the stage S_2 has to perform both the alignment and the instruction tuning. In summary, both S_0 and S_1 contribute to the performance, with S_0 having the larger impact, which shows that the text-only recovery is necessary for retrofitting recurrence into a pruned VLM.

Layer allocation. We first ablate the design of the recurrent block in Table 3(a). To this end, we fix the depth of the recurrent model at m^r=56 layers, thus smaller recurrent blocks are executed with more recurrences compared to larger ones. All models are trained on a $7.5B$ S_0 text-token budget. We find that a larger recurrent block performs better than other variants given the same compute setting. Namely, the $4/8/4$ allocation outperforms a $6/4/6$ allocation at r=11 by 3.19 pp. A narrower $4/6/4$ block at r=8 prunes two additional layers, and the $4/8/4$ configuration outperforms it by 1.60 pp.

Adapters. Next, we consider the adapter design in Table 3(b), comparing the proposed linear adapter of Eq. 3 against multiple adapter variants, including the case of no adapter. The gated adapter A_{sc} adds the context h to the state s, multiplied channel-wise by a learned factor $(1+\alpha)$ with α initialized at zero. Thus the adapter starts as a plain sum: $A_{\text{sc}}(s,h) = s + (1+\alpha) \odot h$. The Parcae adapter A_{pc} follows the adapter design of Prairie et al. (2026). It multiplies the state by a learned per-channel factor $a \in (0,1)^H$, so that the state of the previous recurrence is damped and cannot grow across recurrences, and adds a learned projection W' of the normalized context, weighted by a learned per-channel scale δ: $A_{\text{pc}}(s,h) = a \odot s + \delta \odot W' \,\text{LN}(h)$. The no adapter variant $A_\varnothing$ starts the loop from $s_0 = h$ instead of noise. The evaluation is done on a 4/6/4 architecture with $r = 6$ and all models are trained on a $7.5B$ S_0 text-token budget. Overall, it shows that the linear adapter A_{lin} (Eq. 3) achieves the highest performance. The adapter-free configuration $A_\varnothing$ initializes the loop directly from the prelude output h and noticeably performs nearly as well as A_{lin} with a performance gap of only 0.73 pp. Furthermore, $A_\varnothing$ outperforms both the gated adapter A_{sc} and the Parcae adapter A_{pc} by 1.31 pp and 0.53 pp, respectively.

Looping Variants. We compare block-wise looping with per-layer looping in Table 3(c). Per-layer looping loops each layer of the recurrent block individually (Gao et al., 2026) instead of the block as a whole. Namely, we again consider a 4/6/4 architecture trained on a $7.5B$ S_0 text-token

Table 4: Knowledge distillation. We evaluate the impact of the knowledge distillation in S_2. Each row states on which batches the distillation term is applied and with which weight λ and temperature τ. All rows use the 60B S_0 budget and the $4/8/4$ layer allocation and share the same S_0, which applies the distillation term with λ=0.5 at τ=2. The first two rows give the unpruned baseline of Table 1 with and without the S_2 distillation.

Knowledge distillation	**Avg**
Baseline+(S_0, S_1, S_2) (Table 1)	55.80
without S_2 distillation	55.55
RR-VLM, no S_2 distillation	56.65
RR-VLM, VL-KD only, λ=0.25	56.88
+ text-KD, τ=2	57.24
+ text-KD, τ=1 (ours)	**58.13**

Table 5: Fixed against sampled training depth. Both models use the $4/8/4$ layer allocation and the 7.5B S_0 budget and differ only in the number of recurrences during training, fixed at r=6 or sampled with mean 6. The columns give GQA accuracy when the model is evaluated with r passes. Values marked † come from runs that were stopped early because the output had degenerated.

	GQA at test-time depth r						
Training depth	r=2	r=4	r=5	r=6	r=8	r=12	r=16
sampled, $\bar{r}$=6 ($F^{\tilde{r}}$)	57.8	59.3	59.5	59.5	59.4	59.1	59.3
fixed, r=6 (F^{r}, our setting)	0.0†	0.0†	43.1	59.7	50.9	4.2†	0.0†

budget, but with per-layer looping for r=2 and r=6 recurrences. Overall, it shows that block looping outperforms per-layer looping by 2.83 pp at r=2 and 5.40 pp at r=6 and that per-layer looping does not profit from an increase of recurrences, with the performance dropping with more loops.

Knowledge distillation. We further evaluated the impact of knowledge distillation, especially at the training stage S_2, and compared this setting with no knowledge distillation, a knowledge distillation based on image-text samples only, as well as a knowledge distillation with text-image as well as text-only training data in Table 4. Overall, it shows that knowledge distillation in S_2 always improves the overall performance, and that having both image-text as well as text-only data leads to the most significant improvement, improving over the undistilled version by 1.48 pp. The unpruned baseline gains only 0.25 pp from the same S_2 distillation (first two rows). We attribute the gain of the recurrent model to the fact that the knowledge distillation acts as a form of regularizer for the new recurrent architecture, which keeps the model close to the pretrained one and prevents it from overfitting on the image-text data. This is in line with the large effect of removing S_0 in Table 2, which also anchors the retrofitted model to the pretrained one. Appendix A.2 reports the placement of the distillation term across the training stages at the 7.5B budget.

Fixed versus sampled depth. We further evaluate the effect of the number of recurrences during training in Table 5. To this end, we compare a model trained with a fixed number of recurrences r=6 against a model trained with a variable number of recurrences, sampled per training step with mean 6 (Sec. 3.1). Both models are evaluated at test-time depths from r=2 to r=16. We find that the model trained with a fixed recurrence only works at the depth it was trained for, degrades at the neighbouring depths and collapses beyond r=8 or below r=5. Its GQA accuracy drops from 59.7 at r=6 to 43.1 at r=5 and 50.9 at r=8, and the output degenerates at all other depths. In contrast, the model trained with a sampled recurrence regime also works at depths other than the one it was trained for, staying within 2 pp of its best value from r=2 to r=16. This can be attributed to the fact that the sampled regime exposes the model to a different number of recurrences in every training step, so that its readout is trained for every depth in that range. However, we find that its performance saturates and does not improve beyond the trained depth. Consequently, we do not observe test-time scaling. The fixed and sampled recurrence models perform on par at the trained depth, with 55.75 against 55.83 on Avg. We therefore find that a fixed recurrence model is the less expensive approach to train a recurrent VLM, but it comes at the price of test-time stability to varying recurrences. Appendix A.5 evaluates early exit for the sampled model.

KV-Cache Sharing. Finally, we evaluate the effect of sharing the KV cache across recurrences at inference in Table 6. To this end, we apply the inference-time sharing of Sec. 3.1 to our trained model and reduce the number of cache copies n from one per recurrence, which is our setting, down

Table 6: KV-cache sharing across passes at inference. The table analyses the KV cache per token as a multiple of the base model's cache. The model keeps only n copies of the cache instead of one per pass. The six passes use these copies in turn, so with n=1 every pass shares one cache, n=6 is our model without sharing.

Setting	Cache	GQA	POPE	TVQA	SQA	MMB	RWQA	SEED	AI2D	CV2D	Avg
n=1 (last pass)	0.727×	49.1	86.7	18.6	59.3	57.0	47.7	61.2	42.2	52.5	52.70
n=2	1.09×	56.5	86.7	31.5	59.2	58.6	49.7	62.6	44.1	53.8	55.85
n=3	1.45×	58.6	86.7	41.8	59.4	58.6	50.5	62.6	44.4	53.9	57.38
n=6 (ours)	2.55×	59.2	86.7	47.7	59.1	58.9	50.5	62.6	44.6	53.9	58.13

to a single copy. Since every pass through the recurrent block keeps its own keys and values, our model holds 2.55× the KV cache of the base model, in line with the 56 against 22 traversed layers per token, while it stores 18.1% fewer weights and outperforms it (Table 1). We see that the cost grows as the budget n shrinks. With n=3 the cache shrinks to 1.45× that of the base model at a cost of 0.75 pp, and with n=1 it shrinks to 0.727×, below the base model itself, at a cost of 5.43 pp. This can be attributed to the fact that the keys and values of a recurrence step are computed from the state of that step, so a recurrence step that reads the cache of another step attends to representations that do not match its own input. Training the model with the sharing active does not close this gap either, with a cost of 0.73 pp for sharing the keys and 2.94 pp for sharing the keys and values (Appendix A.3). We thus keep a separate cache per recurrence in our best-performing setting, and note that the cache size can be traded against accuracy at inference without retraining. Table 8 in the appendix reports sharing applied during training.

5.3 Qualitative examples

Figure 2 shows one item from each of the four reasoning-oriented benchmarks on which our model answers correctly and all three trained non-recurrent baselines of Table 1 answer wrongly. We compare our model to the pretrained model and to the converted recurrent VLM, which is our closest recurrent baseline. For further qualitative examples, we refer to Appendix A.6.



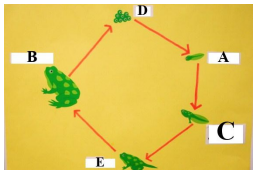
RealWorldQA
Where is the orange ball located relative to the white dog? A. left of the dog B. right of the dog C. behind the dog

TinyLLaVA-1.4B	C ✗
Recurrent LM+ViT	A ✓
RR-VLM (ours)	A ✓



SEED-Bench
How many pillows are on the bed closest to the window? A. 0 B. 1 C. 3 D. 2

TinyLLaVA-1.4B	D ✗
Recurrent LM+ViT	D ✗
RR-VLM (ours)	B ✓



AI2D
What stage is between a tadpole with legs and an adult frog? A. adult frog B. tadpole C. egg D. froglet

TinyLLaVA-1.4B	B ✗
Recurrent LM+ViT	B ✗
RR-VLM (ours)	D ✓



CV-Bench-2D
Considering the relative positions of the flowerpot and the picture, where is the flowerpot located with respect to the picture? A. left B. right

TinyLLaVA-1.4B	B ✓
Recurrent LM+ViT	A ✗
RR-VLM (ours)	B ✓

Figure 2: Qualitative examples. One item from each of the four reasoning-oriented benchmarks with the answers of the pretrained model, the converted recurrent model and our model below the image, a check mark denotes the correct option. The three trained non-recurrent baselines of Table 1 answer all four items wrongly.

6 Conclusion

We introduced the first prune-then-loop retrofit of a pretrained VLM evaluated on VQA. The retrofitted model outperforms the unpruned baseline while storing 23.3% fewer language-model parameters, with the gains concentrated on reasoning-oriented benchmarks. This gain is due to the recurrence itself, as the loop accounts for a significant improvement over the pruned single-pass baseline. At the same time, the recovery requires the text-only training stage, which yields the best performance among all evaluated training strategies. Finally, retrofitting recurrence into a pretrained VLM outperforms adding a vision encoder to a pretrained recurrent language model, which makes the retrofit the best performing route to a recurrent VLM among all evaluated baselines.

Reproducibility statement

Appendix A.1 lists every training hyperparameter of every model in the paper. We will release the code and the checkpoints.

AI use statement

Generative-AI assistants were used to polish the writing and to implement the baselines and the proposed method. The experiments, the results and their interpretation are the authors' own.

References

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

Lizhang Chen, Jonathan Li, Chen Liang, Ni Lao, and Qiang Liu. Training-free looped transformers. *arXiv preprint arXiv:2605.23872*, 2026.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019.

Haodong Duan, Junming Yang, Yuxuan Qiao, Xinyu Fang, Lin Chen, Yuan Liu, Xiaoyi Dong, Yuhang Zang, Pan Zhang, Jiaqi Wang, Dahua Lin, and Kai Chen. VLMEvalKit: An open-source toolkit for evaluating large multi-modality models. In *ACM International Conference on Multimedia*, 2024.

Zitian Gao, Yilong Chen, Yihao Xiao, Xinyu Yang, Ran Tao, Joey Zhou, and Bryan Dai. Loop the loopies! *arXiv preprint arXiv:2607.16051*, 2026.

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. *arXiv preprint arXiv:2502.05171*, 2025.

Yiran Huang, Lukas Thede, Massimiliano Mancini, Wenjia Xu, and Zeynep Akata. Structural pruning of large vision language models: A comprehensive study on pruning dynamics, recovery, and data efficiency. *International Journal of Computer Vision*, 2026.

Drew A. Hudson and Christopher D. Manning. GQA: A new dataset for real-world visual reasoning and compositional question answering. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks. `https://kellerjordan.github.io/posts/muon/`, 2024.

Aniruddha Kembhavi, Mike Salvato, Eric Kolve, Minjoon Seo, Hannaneh Hajishirzi, and Ali Farhadi. A diagram is worth a dozen images. In *European Conference on Computer Vision*, 2016.

Yeskendir Koishekenov, Aldo Lipani, and Nicola Cancedda. Encode, think, decode: Scaling test-time reasoning with recursive latent thoughts. *arXiv preprint arXiv:2510.07358*, 2025.

Dohyeon Kwon and Youngjin Park. CHERRY: Compressed hierarchical experts with recurrent representational yield. *arXiv preprint arXiv:2606.31796*, 2026.

Bohao Li, Rui Wang, Guangzhi Wang, Yuying Ge, Yixiao Ge, and Ying Shan. SEED-Bench: Benchmarking multimodal LLMs with generative comprehension. *arXiv preprint arXiv:2307.16125*, 2023a.

Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. BLIP: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International Conference on Machine Learning*, 2022.

Yifan Li, Yifan Du, Kun Zhou, Jinpeng Wang, Wayne Xin Zhao, and Ji-Rong Wen. Evaluating object hallucination in large vision-language models. In *Conference on Empirical Methods in Natural Language Processing*, 2023b.

Ziyue Li, Yang Li, and Tianyi Zhou. Skip a layer or loop it? learning program-of-layers in LLMs. In *International Conference on Machine Learning*, 2026.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *Advances in Neural Information Processing Systems*, 2023.

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 *European Conference on Computer Vision*, 2024.

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

Pan Lu, Swaroop Mishra, Tony Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. In *Advances in Neural Information Processing Systems*, 2022.

Ji Ma, Wei Suo, Peng Wang, and Yanning Zhang. Short-LVLM: Compressing and accelerating large vision-language models by pruning redundant layers. In *ACM International Conference on Multimedia*, 2025.

Parsa Madinei, Ryan Solgi, Ziqi Wen, Jonathan Skaza, Miguel Eckstein, and Ramtin Pedarsani. INTERLACE: Interleaved layer pruning and efficient adaptation in large vision-language models. *arXiv preprint arXiv:2511.19676*, 2025.

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. *arXiv preprint arXiv:2511.07384*, 2025.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2024.

Hayden Prairie, Zachary Novack, Taylor Berg-Kirkpatrick, and Daniel Y. Fu. Parcae: Scaling laws for stable looped language models. *arXiv preprint arXiv:2604.12946*, 2026.

Yiming Qin, Bomin Wei, Jiaxin Ge, Konstantinos Kallidromitis, Stephanie Fu, Trevor Darrell, and XuDong Wang. Chain-of-visual-thought: Teaching VLMs to see and think better with continuous visual tokens. *arXiv preprint arXiv:2511.19418*, 2025.

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 *International Conference on Machine Learning*, 2021.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In *Advances in Neural Information Processing Systems*, 2022.

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.

Boyang Shen, Kaixiang Yang, Hao Wang, Qiuyu Yu, Qiang Xie, Qiang Li, and Zhiwei Wang. LoopVLA: Learning sufficiency in recurrent refinement for vision-language-action models. *arXiv preprint arXiv:2605.09948*, 2026.

Akshit Singh, Shyam Marjit, Wei Lin, Paul Gavrikov, Serena Yeung-Levy, Hilde Kuehne, Rogerio Feris, Sivan Doveh, James Glass, and M. Jehanzeb Mirza. TTRV: Test-time reinforcement learning for vision language models. *arXiv preprint arXiv:2510.06783*, 2025.

Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. Towards VQA models that can read. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R. Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627b token cleaned and deduplicated version of RedPajama. `https://cerebras.ai/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama`, 2023.

Shengbang Tong, Ellis Brown, Penghao Wu, Sanghyun Woo, Manoj Middepogu, Sai Charitha Akula, Jihan Yang, Shusheng Yang, Adithya Iyer, Xichen Pan, Ziteng Wang, Rob Fergus, Yann LeCun, and Saining Xie. Cambrian-1: A fully open, vision-centric exploration of multimodal LLMs. In *Advances in Neural Information Processing Systems*, 2024.

Yalcin Tur, Jalal Naghiyev, Haoquan Fang, Wei-Chuan Tsai, Jiafei Duan, Dieter Fox, and Ranjay Krishna. Recurrent-depth VLA: Implicit test-time compute scaling of vision–language–action models via latent iterative reasoning. *arXiv preprint arXiv:2602.07845*, 2026.

Cong Wan, Ying He, Zhongzhan Huang, and Hefeng Wu. Test-time scaling in multimodal foundation models: A comprehensive survey of generation and reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026.

Shijie Wang, Xiangzhao Hao, Yueti Li, Guangyu Cao, Xinyu Tang, and Haiyun Guo. ReLoopUME: Recurrent depth with learnable retrieval registers for universal multimodal embedding. *arXiv preprint arXiv:2607.28751*, 2026.

xAI. Grok-1.5 vision preview. `https://x.ai/news/grok-1.5v`, 2024.

Guowei Xu, Peng Jin, Ziang Wu, Hao Li, Yibing Song, Lichao Sun, and Li Yuan. LLaVA-CoT: Let vision language models reason step-by-step. In *IEEE/CVF International Conference on Computer Vision*, 2025.

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*, 2026.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. TinyLlama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024a.

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

Zhuosheng Zhang, Aston Zhang, Mu Li, Hai Zhao, George Karypis, and Alex Smola. Multimodal chain-of-thought reasoning in language models. *Transactions on Machine Learning Research*, 2024b.

Baichuan Zhou, Ying Hu, Xi Weng, Junlong Jia, Jie Luo, Xien Liu, Ji Wu, and Lei Huang. TinyLLaVA: A framework of small-scale large multimodal models. *arXiv preprint arXiv:2402.14289*, 2024.

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 Appendix

A.1 Training hyperparameters

Every stage uses Muon (Jordan et al., 2024) for the matrix parameters and AdamW for all other parameters, with weight decay 10^{-4}, gradient clipping at 1.0 and a warmup-stable-decay schedule with 5% warmup. In S_0 the learning rates are $5{\cdot}10^{-4}$ for Muon and $2.5{\cdot}10^{-5}$ for AdamW, the batch holds 128 sequences of 2,048 tokens, and the learning rate decays to zero over the last 35% of the steps. The 60B run of our model takes 228,874 steps on 16 GPUs, and the 7.5B run of the ablations takes 28,611 steps. In S_1 and S_2 the learning rates are 10^{-3} for Muon and $5{\cdot}10^{-5}$ for AdamW, the learning rate decays to zero over the last 60% of the steps, and both stages run on 4 GPUs. S_1 trains the projector for 6,426 steps with a batch of 96 image-text pairs, and S_2 trains the whole model for 6,408 steps with a batch of 128, one pass over the data in both stages with the interleaved text batches included in the step counts. The distillation weights and temperatures of each stage are given in Sec. 4. The Parcae adapter A_{pc} of Table 3 uses an additional learning rate of 10^{-3} for its adapter parameters, and the per-layer looping variants of Table 3(c) share the recipe of the 7.5B cells. The converted recurrent VLM keeps the 4/8/4 geometry of our model and is trained with its own trainer with the same learning rates and a batch of 128 in both stages.

A.2 Distillation placement at the 7.5B budget

Table 7 applies the distillation term of Eq. 4 in the stages S_0, S_1 and S_2, in S_0 only, or in S_0 and S_2 with weight $\lambda{=}0.50$ or $\lambda{=}0.25$, on the model with the $4/6/4$ layer allocation and $r{=}6$ at the 7.5B S_0 budget.

Table 7: Placement of the distillation term at the 7.5B S_0 budget. Each row shows in which training stages we apply the distillation term and with which weight λ, on the $4/6/4$ layer allocation with $r{=}6$. Seed 42. These are separate runs from the $4/6/4$ row of Table 3(b).

Distillation	Avg
S_0, S_1, S_2 (every stage)	53.53
S_0 only	54.57
S_0, S_2, $\lambda{=}0.50$	54.45
S_0, S_2, $\lambda{=}0.25$	**55.62**

A.3 KV-cache sharing during training

Table 8 trains the model with the $4/8/4$ layer allocation of our ablations at the 7.5B S_0 budget with the keys, or the keys and values, shared across passes (Sec. 3.1).

Table 8: KV-cache sharing across passes during training. The model is trained with the sharing active, at the 7.5B S_0 budget and the $4/8/4$ layer allocation of our ablations. The Cache column gives the KV cache per token as a multiple of the base model's cache. K-sharing shares the keys across passes, KV-sharing the keys and the values. Seed 42.

Setting	Cache	GQA	POPE	TVQA	SQA	MMB	RWQA	SEED	AI2D	CV2D	Avg
KV-sharing	$0.727\times$	57.6	86.1	44.5	51.1	50.0	43.8	57.8	37.3	47.1	52.81
K-sharing	$1.64\times$	58.5	86.6	44.7	53.5	56.4	48.5	61.8	39.1	46.1	55.02
no sharing	$2.55\times$	59.7	86.5	45.1	57.3	55.1	46.7	62.8	41.1	47.5	55.75

A.4 Text interleaving in S_2

Stage S_2 interleaves one text-only batch after every K multimodal batches. Table 9 varies K on the 60B model without text distillation, which is the VL-KD only row of Table 4. Removing the text batches costs 0.69 pp, and $K{=}2$ or $K{=}6$ cost 0.20 pp and 0.07 pp relative to $K{=}4$.

Table 9: Text interleaving in S_2. Number of multimodal batches K per text batch during S_2, on the 60B S_0 budget and the $4/8/4$ layer allocation without text distillation. The best value is in bold. Seed 42.

Interleaving	Avg
no text batches	56.19
K=2	56.68
K=6	56.81
K=4 (our setting)	**56.88**

A.5 EARLY EXIT

To reduce inference cost, the step-wise generation of output logits (Eq. 3) enables an adaptive halting criterion (Schuster et al., 2022; Geiping et al., 2025) that terminates the forward pass dynamically at step

$$k^\star \;=\; \min\{\, k \geq 2 : \mathrm{KL}\big(\,\mathrm{softmax}(y_k)\;\big\|\;\mathrm{softmax}(y_{k-1})\big) < \epsilon \,\}, \tag{5}$$

where the threshold ϵ defines the accuracy-compute tradeoff. In the read-out-only form every pass is still computed and the token is read from pass $k^\star$. In the compute-skipped form the passes after $k^\star$ are not computed. Table 10 reports both forms on GQA, each against the same model without exit (Δ).

Table 10: Early exit (Eq. 5) on GQA. The retrofits use the 7.5B-token S_0 and are trained at a fixed depth of 6 or with depth sampled at mean 6 or 16. The converted model at mean depth 32 is built like the converted recurrent VLM of Table 1 from the McLeish et al. (2025) checkpoint pretrained at that depth. Pass read is the mean pass a generated token is read from. Passes run is the mean number of block passes computed per generated token, with the first token counted at the cap (†: tokens after the first only). Settled@6 is the share of generated tokens whose successive readouts have a KL divergence below $5{\cdot}10^{-3}$ at pass 6. Rows without exit have no ϵ and no Δ. The readouts of the fixed-depth model do not settle, so the rule almost never fires on it (pass read 6.00 and 5.92). The values without exit differ from Tables 1 and 5 because this evaluation ran without image padding.

Model	Variant	ϵ	Cap	GQA	Δ	Pass read	Passes run	Settled @6
sampled, mean 6	no exit	–	6	58.92	–	6.00	6.00	98.4%
	read-out only	$5{\cdot}10^{-3}$	6	58.53	-0.39	2.81	6.00	
	compute-skipped	$5{\cdot}10^{-3}$	6	58.31	-0.61	2.79	3.75	
	compute-skipped	10^{-3}	6	58.76	-0.16	3.40	3.89	
fixed, 6	no exit	–	6	59.41	–	6.00	6.00	3.7%
	read-out only	$5{\cdot}10^{-3}$	6	59.41	$+0.00$	6.00	6.00	
	compute-skipped	$5{\cdot}10^{-3}$	6	59.41	$+0.00$	5.92	5.92	
sampled, mean 16	no exit	–	16	59.06	–	16.00	16.00	–
	read-out only	$5{\cdot}10^{-3}$	16	58.40	-0.67	3.15	16.00	
	compute-skipped	$5{\cdot}10^{-3}$	16	58.55	-0.52	$2.28^\dagger$	$2.28^\dagger$	
	no exit	–	32	59.12	–	32.00	32.00	
	compute-skipped	$5{\cdot}10^{-3}$	32	58.71	-0.41	$2.28^\dagger$	$2.28^\dagger$	
converted, 8	no exit	–	8	58.36	–	8.00	8.00	71.4%
	read-out only	$5{\cdot}10^{-3}$	8	57.40	-0.96	3.77	8.00	
	compute-skipped	$5{\cdot}10^{-3}$	8	57.39	-0.98	3.72	4.72	
converted, mean 32	no exit	–	32	59.04	–	32.00	32.00	77.3%
	read-out only	$5{\cdot}10^{-3}$	32	58.27	-0.77	3.97	32.00	
	compute-skipped	$5{\cdot}10^{-3}$	32	58.20	-0.83	3.93	14.70	

A.6 QUALITATIVE EXAMPLES

Figures 3 and 4 show two further items from each of the four reasoning-oriented benchmarks on which our model answers correctly and the three trained non-recurrent baselines of Table 1, Baseline+(S_1, S_2), Baseline+(S_0, S_1, S_2) and Baseline(pruned)+(S_0, S_1, S_2), answer wrongly, together with the answer of every model in Table 1. A check mark denotes the correct option.

	**RealWorldQA** Given the picture, in which cardinal direction is the dinosaur facing? A. North. B. South. C. East. D. West	TinyLLaVA-1.4B Baseline+(S_1, S_2) Baseline+(S_0, S_1, S_2) Baseline(pruned)+(S_0, S_1, S_2) Recurrent LM+ViT+(S_1, S_2) RR-VLM (ours)	D × A × A × D × C ✓ C ✓
	**RealWorldQA** Which direction is the blue cap pens arrow pointing? A. Pointing up. B. Pointing to the right. C. Pointing to the left.	TinyLLaVA-1.4B Baseline+(S_1, S_2) Baseline+(S_0, S_1, S_2) Baseline(pruned)+(S_0, S_1, S_2) Recurrent LM+ViT+(S_1, S_2) RR-VLM (ours)	C × C × C × C × C × B ✓
	**SEED-Bench** What is the relative position between the leaf and the flower? A. The leaf is above the flower B. The leaf is on the right of the flower C. The leaf is on the left of the flower D. The leaf is below the flower	TinyLLaVA-1.4B Baseline+(S_1, S_2) Baseline+(S_0, S_1, S_2) Baseline(pruned)+(S_0, S_1, S_2) Recurrent LM+ViT+(S_1, S_2) RR-VLM (ours)	A × A × A × A × A × D ✓
	**SEED-Bench** Is the laptop above, below, or next to the family in the image? A. Next to B. Below C. Above D. In front of	TinyLLaVA-1.4B Baseline+(S_1, S_2) Baseline+(S_0, S_1, S_2) Baseline(pruned)+(S_0, S_1, S_2) Recurrent LM+ViT+(S_1, S_2) RR-VLM (ours)	D ✓ C × C × C × D ✓ D ✓

Figure 3: Qualitative examples (RealWorldQA, SEED-Bench). Two items per benchmark with the answer of every model in Table 1. A check mark denotes the correct option.

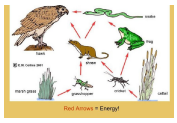
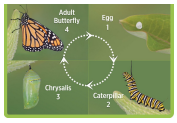
	**AI2D** Which is a secondary consumer? A. shrew B. grasshopper C. cricket D. grass	TinyLLaVA-1.4B Baseline+(S_1, S_2) Baseline+(S_0, S_1, S_2) Baseline(pruned)+(S_0, S_1, S_2) Recurrent LM+ViT+(S_1, S_2) RR-VLM (ours)	B × B × B × B × B × A ✓
	**AI2D** What does stage the number 3 depict? A. Adult B. Egg C. Chrysalis D. Caterpillar	TinyLLaVA-1.4B Baseline+(S_1, S_2) Baseline+(S_0, S_1, S_2) Baseline(pruned)+(S_0, S_1, S_2) Recurrent LM+ViT+(S_1, S_2) RR-VLM (ours)	B × B × B × D × B × C ✓
	**CV-Bench-2D** Considering the relative positions of the person (annotated by the red box) and the horse in the image provided, where is the person (annotated by the red box) located with respect to the horse? A. left B. right	TinyLLaVA-1.4B Baseline+(S_1, S_2) Baseline+(S_0, S_1, S_2) Baseline(pruned)+(S_0, S_1, S_2) Recurrent LM+ViT+(S_1, S_2) RR-VLM (ours)	A × A × A × A × A × B ✓
	**CV-Bench-2D** Considering the relative positions of the cutlery and the range in the image provided, where is the cutlery located with respect to the range? A. left B. right	TinyLLaVA-1.4B Baseline+(S_1, S_2) Baseline+(S_0, S_1, S_2) Baseline(pruned)+(S_0, S_1, S_2) Recurrent LM+ViT+(S_1, S_2) RR-VLM (ours)	A × A × A × A × A × B ✓

Figure 4: Qualitative examples (AI2D, CV-Bench-2D). Two items per benchmark with the answer of every model in Table 1. A check mark denotes the correct option.