AR-RETROFIT: RETROFITTING PRETRAINED DECODERS WITH ATTENTION RESIDUALS FOR ADAPTIVE DEPTH

Anonymous authors
Paper under double-blind review

ABSTRACT

Attention Residuals improve Transformer performance by enabling adaptive aggregation of representations across different depths. However, applying this mechanism to existing pretrained models remains challenging, as direct integration requires expensive backbone retraining and introduces additional inference overhead. To avoid costly backbone retraining, we propose Attention Residual Retrofit (AR-Retrofit), a lightweight framework for frozen pretrained decoders. AR-Retrofit introduces depth aggregation through gated residual corrections while preserving the original forward function at initialization. The magnitude of these corrections is adaptively controlled by Residual Strength Modulation for different tokens and depths. To reduce the additional inference overhead, we propose Routing-guided Residual Skipping (ReSkip), which reuses the routing signals learned by AR-Retrofit for selective layer execution without training an additional controller. Experiments on language, multimodal, and robotic tasks show that AR-Retrofit transfers adaptive depth aggregation to pretrained models while improving performance across all three domains. On Qwen3-VL-2B, it adds less than 0.4% trainable parameters and requires only 12.6 minutes of training on a single H100. Across six VLM benchmarks, average scores improve by 2.08 and 1.44 points for Qwen3-VL-2B and 4B, respectively. ReSkip increases decoding throughput over Base by 4.5%–8.8% while retaining most gains. These results show that our framework enables lightweight and efficient adaptive depth in pretrained models.

1 INTRODUCTION

Different inputs do not always require the same amount of computation. Studies of human perception suggest a similar pattern: simple stimuli can often be resolved through rapid feedforward processing, whereas complex or ambiguous inputs require deeper recurrent computation (Kahneman, 2011; Lamme & Roelfsema, 2000; DiCarlo et al., 2012; Kar et al., 2019; Kietzmann et al., 2019; Spoerer et al., 2020). Modern large models already adapt computation along the token dimension. Scaling computation at test time, chain-of-thought prompting, and methods that generate intermediate thoughts allocate more computation to difficult problems by producing additional intermediate tokens (OpenAI, 2024; DeepSeek-AI, 2025; Snell et al., 2024; Wei et al., 2022; Zelikman et al., 2024). Yet each token still passes through a fixed sequence of Transformer blocks. Depth therefore remains a complementary axis for adaptive computation in LLMs, VLMs, and VLA models.

Existing adaptive depth methods reduce computation through early exit (Elhoushi et al., 2024), routing decisions for individual tokens (Raposo et al., 2024; Luo et al., 2025), or layer pruning after training (Gromov et al., 2025). Another line of work changes how representations are combined across depth by learning connections between layers and mixing residual streams (Zhu et al., 2025; Xie et al., 2026). Attention Residuals (AttnRes) assign content-dependent weights to earlier layer or block outputs instead of accumulating them with unit weights. Each block can then reuse representations from different depths, with the aggregation weights also serving as routing signals. Pretraining experiments demonstrate the effectiveness of this aggregation scheme (Kimi Team et al., 2026).

Retrofitting AttnRes into a pretrained decoder raises three challenges. (i) Preserving pretrained behavior. Directly replacing the residual path of an existing model changes the representations

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.

received by downstream blocks, while retraining the pretrained backbone is costly. (ii) Controlling the correction strength. Routing determines how historical representations are combined, but it does not determine how strongly the resulting correction should affect the input to a frozen block. This strength may need to vary across tokens and depths. (iii) Deployment efficiency. The added routing and adaptation modules introduce extra computation, while representation aggregation alone does not reduce the number of executed layers. A practical retrofit therefore needs to offset this overhead without introducing another costly controller.

We propose Attention Residual Retrofit (AR-Retrofit), a lightweight framework that brings AttnRes aggregation to frozen pretrained decoders. Instead of replacing the original residual path, AR-Retrofit uses the difference between the routed state and the predecessor state as a residual correction. A small adapter transforms this correction, while a γ schedule starting at zero gradually introduces it into the backbone. Residual Strength Modulation (RSM) adjusts its magnitude for each token and depth. On Qwen3-VL-2B, the added trainable parameters account for less than 0.4% of the model.

To reduce inference overhead, we propose Routing-guided Residual Skipping (ReSkip). ReSkip reuses the routing signals from AR-Retrofit to decide which layer computations can be omitted during inference. The routing weights for a target block depend only on completed block states, so they are available before the block executes. ReSkip uses offline sensitivity analysis to select positions eligible for skipping and calibrate decision thresholds. During decoding, these calibrated rules determine which layer computations to skip for each token. After AR-Retrofit adaptation, ReSkip requires no further training or dedicated skip predictor.

We evaluate AR-Retrofit and ReSkip across language, multimodal, and robotic tasks, primarily using Qwen3-VL-2B and Qwen3-VL-4B. AR-Retrofit improves language and multimodal performance and provides a stronger initialization for LIBERO policies, which outperform the corresponding Base policies. On VLM tasks, ReSkip reduces decode latency by 6.0% to 9.7% relative to AR-Retrofit and increases decode throughput by 4.5% to 8.8% over Base. Its average score remains within 0.4 points of AR-Retrofit. Our main contributions are:

- **AR-Retrofit.** We introduce a lightweight framework to bring AttnRes depth aggregation into frozen pretrained decoders. AR-Retrofit preserves the original forward behavior at initialization and gradually learns to incorporate historical block representations through residual corrections, with RSM controlling the injection strength across tokens and depths. AR-Retrofit requires only 12.6 minutes on a single H100 for Qwen3-VL-2B adaptation, avoiding the need for costly backbone retraining.
- **ReSkip.** Building on the routing signals learned by AR-Retrofit, we propose ReSkip to extend adaptive depth from representation aggregation to selective execution. Without additional training or a separate controller, ReSkip selectively skips layer computations through offline calibration while preserving most of AR-Retrofit's performance gains.
- **Evaluation across domains.** We evaluate the proposed framework across language, multimodal understanding, and robotic manipulation tasks. AR-Retrofit improves average performance across all three domains, including 1.44–2.08 point gains on six VLM benchmarks with Qwen3-VL-2B and 4B. ReSkip fully offsets the additional inference overhead introduced by adaptive depth and achieves 4.5%–8.8% higher decoding throughput than Base on VLM benchmarks while retaining most performance gains.

2 Related Work

Adaptive computation and depth. Reasoning methods use longer token sequences, repeated sampling, or search, typically with fixed depth per token (OpenAI, 2024; DeepSeek-AI, 2025; Wei et al., 2022; Snell et al., 2024). Stochastic Depth drops layers during training (Huang et al., 2016). ACT, Universal Transformers, and PonderNet adapt the number of recurrent computation steps (Graves, 2016; Dehghani et al., 2019; Banino et al., 2021); recurrent language models also scale latent reasoning by repeating a shared block (Geiping et al., 2025). Depth-Adaptive Transformer and CALM enable early exit (Elbayad et al., 2020; Schuster et al., 2022), Mixture-of-Depths routes tokens across layers (Raposo et al., 2024), and LayerSkip combines early exit with self-speculative decoding (Elhoushi et al., 2024). Static pruning methods remove layers after training (Gromov et al., 2025; Men et al., 2025; He et al., 2026). FlexiDepth trains routers and adapters for selective layer

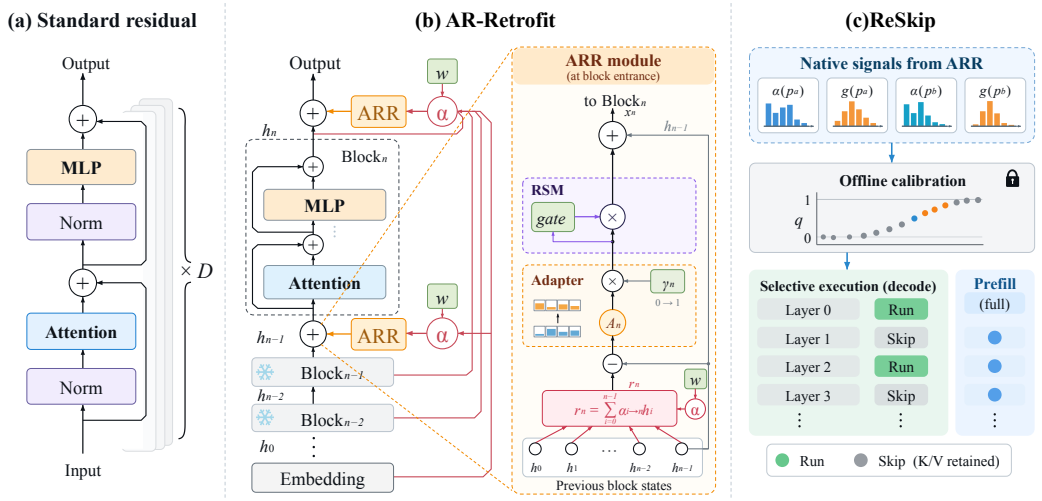


Figure 1: Overview of AR-Retrofit (ARR) and ReSkip. (a) A standard residual decoder follows a fixed computation path. (b) AR-Retrofit aggregates previous block states and injects a correction $g_n u_n$ scaled by RSM into the frozen backbone. All blocks execute during adaptation. (c) After offline calibration, ReSkip reuses native signals at each block entrance to decide whether to omit a fixed subset of decoder layers for the current token during decoding. The layers inside each block retain standard residual connections.

execution in a frozen backbone (Luo et al., 2025). Dr.LLM trains layer routers using supervision from offline search (Heakl et al., 2026), while CoLa searches for execution sequences at test time (Li et al., 2025). ReSkip reuses routing signals from representation aggregation for selective execution after offline calibration, without training a separate controller.

Residual pathways and depth aggregation. DenseNet, Highway Networks, Hyper-Connections, and mHC modify connections across layers or residual streams (Huang et al., 2017; Srivastava et al., 2015; Zhu et al., 2025; Xie et al., 2026). AttnRes learns attention weights over earlier layer outputs and was evaluated through model pretraining (Kimi Team et al., 2026). AR-Retrofit brings this form of aggregation to frozen pretrained decoders.

Parameter efficient adaptation. Adapters add trainable modules to a frozen backbone (Houlsby et al., 2019), while LoRA and DoRA parameterize updates to its weights (Hu et al., 2022; Liu et al., 2024a). Prefix tuning learns continuous prefixes for attention (Li & Liang, 2021), and $(\mathrm{IA})^3$ learns vectors that rescale internal activations (Liu et al., 2022). AR-Retrofit also trains a small set of added parameters, but constructs its residual corrections by explicitly aggregating states from earlier blocks. These corrections preserve the original forward function at initialization.

3 Method

Our method first adapts information flow across depth and then introduces selective execution. AR-Retrofit learns aggregation across blocks and Residual Strength Modulation (RSM) in a frozen pretrained decoder. ReSkip then reuses these signals at block entrances to control the selective execution of predefined layer subsets. We train the added modules for representation adaptation, then calibrate selective execution to balance computation savings and changes in model predictions.

3.1 Preliminaries: Attention Residuals

A standard Transformer uses residual connections with unit weights, $x_{\ell+1} = x_\ell + f_\ell(x_\ell)$, where x_ℓ is the input to layer ℓ and f_ℓ is its transformation. Attention Residuals (AttnRes) aggregate historical residual contributions through learned attention over depth (Kimi Team et al., 2026). Each target layer ℓ has a learned pseudo-query $w_\ell \in \mathbb{R}^d$, where d is the hidden dimension. Let v_0 be the input

embedding and $v_i = f_i(x_i)$ the residual branch output of layer i. The target layer input is

$$\alpha_{i\to\ell} = \frac{\exp\big(w_\ell^\top \operatorname{RMSNorm}(v_i)\big)}{\sum_{j=0}^{\ell-1}\exp\big(w_\ell^\top \operatorname{RMSNorm}(v_j)\big)}, \qquad x_\ell = \sum_{i=0}^{\ell-1}\alpha_{i\to\ell} v_i. \tag{1}$$

Here $\alpha_{i\to\ell}$ is the relative weight assigned by the target layer to historical source i. Constructing the input to layer ℓ requires only $\{v_i\}_{i=0}^{\ell-1}$ and w_ℓ, all of which are available before f_ℓ executes.

Block AttnRes groups layers and uses the accumulated residual contributions within each group, together with the partial sum already formed in the current group, as attention sources (Kimi Team et al., 2026). Grouping the layers into N blocks and applying softmax over block representations defines the granularity of depth routing used in this paper. We call routing and input construction at a block entrance stage 1, and computation within the target block stage 2. Selective execution decisions use the routing signals from stage 1.

3.2 AR-RETROFIT: ADDING DEPTH ROUTING TO FROZEN DECODERS

We retrofit a pretrained Transformer with standard residual connections and frozen weights. The added path must preserve the original forward function at initialization, introduce depth routing into representation updates during adaptation, and provide adapted block inputs for selective execution. Local adapters (Houlsby et al., 2019) and low rank adaptation (Hu et al., 2022) do not explicitly aggregate historical block states. A router that only observes and scores states does not change the forward representations. Directly replacing a block input with an AttnRes mixture generally fails to preserve the original function at initialization. AR-Retrofit retains the original predecessor path and introduces depth routing as a learned residual correction to that path.

We partition a decoder with D layers into $N = D/L_{\mathrm{blk}}$ contiguous blocks of L_{blk} layers each. The added path operates only at block boundaries. Attention, feedforward transformations, and standard residual connections within each block remain unchanged. Let x_n denote the input to block $n \in \{1,\dots,N\}$ after residual correction, h_n its actual output state, and h_0 the embedding stream. Under full execution, $h_n = \mathrm{Block}_n(x_n)$; selective execution follows Section 3.3. Each block has a depth routing query, a bottleneck adapter A_n, and a deterministic scalar gate γ_n. RSM is added to all but the first block. We number blocks starting from one in the method, whereas the implementation block IDs in the experimental tables start from zero. The formulas below apply to a single token, with the position index p omitted.

Given completed block states $\{h_i\}_{i=0}^{n-1}$, the router computes

$$\alpha_{i\to n} = \operatorname{softmax}_{i<n}\left(\frac{w_n^\top[\operatorname{RMSNorm}(h_i) + b_i]}{\sqrt{d}}\right), \qquad r_n = \sum_{i=0}^{n-1}\alpha_{i\to n} h_i. \tag{2}$$

Here $w_n \in \mathbb{R}^d$ is a learned pseudo query for the target block, and $b_i \in \mathbb{R}^d$ is a learned source bias. The bias provides a score for each source depth that is independent of content.

The routed state r_n enters the frozen backbone through its deviation from the ordinary predecessor state:

$$\delta_n = r_n - h_{n-1} = \sum_{i=0}^{n-2}\alpha_{i\to n}(h_i - h_{n-1}). \tag{3}$$

The second equality follows from $\sum_{i=0}^{n-1}\alpha_{i\to n} = 1$. This construction retains h_{n-1} as the reference state, so the correction depends only on differences between historical sources and the predecessor state. The bottleneck adapter $A_n(u) = W_n^{\mathrm{up}}\sigma(W_n^{\mathrm{down}}u)$ applies a nonlinear transformation to this deviation. Here $\sigma = \mathrm{SiLU}$ and r is the bottleneck dimension. The adapter satisfies $A_n(0) = 0$.

The routing weights determine the relative combination of historical information, the deterministic gate controls how the correction is introduced during training, and RSM adjusts its strength for the current token. Let $u_n = \gamma_n A_n(\delta_n)$. Under full execution, $h_n = \mathrm{Block}_n(x_n)$, where

$$g_n = 1 + 0.5\tanh\left(\frac{(w_n^{\mathrm{RSM}})^\top \operatorname{RMSNorm}(u_n) + b_n^{\mathrm{RSM}}}{\sqrt{d}}\right), \qquad x_n = h_{n-1} + g_n u_n. \tag{4}$$

The parameters $w_n^{\mathrm{RSM}} \in \mathbb{R}^d$ and $b_n^{\mathrm{RSM}} \in \mathbb{R}$ are learned, and $0.5 < g_n < 1.5$. RSM changes the magnitude of the correction without changing its direction. The first block has only the embedding source, so $r_1 = h_0$, $\delta_1 = 0$, and hence $x_1 = h_0$. We set $g_1 = 1$ and do not install RSM in this block. The router, adapter, and RSM at each block entrance remain active during selective execution.

When $\gamma_n(0) = 0$, the retrofitted model has the same forward function as the pretrained model at initialization (proof in Appendix A1.1). Training uses a deterministic gate schedule $\gamma_n(t)$ that increases linearly from zero to one. The gates are not optimized by gradient descent.

The relation between residual correction and attention over depth follows directly from Eq. 3. Setting $A_n = \mathrm{Id}$ and $g_n = 1$ gives $x_n = h_{n-1} + \gamma_n(r_n - h_{n-1})$, or equivalently

$$x_n = (1-\gamma_n)h_{n-1} + \gamma_n r_n = \sum_{i=0}^{n-2} \gamma_n \alpha_{i\to n} h_i + (1 - \gamma_n + \gamma_n \alpha_{n-1\to n})\, h_{n-1}. \tag{5}$$

For $0 \le \gamma_n \le 1$, these coefficients are nonnegative and sum to $1 - \gamma_n + \gamma_n \sum_i \alpha_{i\to n} = 1$. Thus $\gamma_n = 0$ recovers the ordinary predecessor input, while $\gamma_n = 1$ exactly recovers the attention mixture $x_n = r_n$ over the same set of historical states. Equation 4 also gives

$$x_n - r_n = \gamma_n g_n A_n(\delta_n) - \delta_n. \tag{6}$$

Our method retains the depth routing weights of AttnRes, learns an output subspace for the correction through the adapter, and uses RSM to adjust the injection magnitude.

Training objective. We freeze the pretrained backbone and optimize only the routers, bottleneck adapters, and RSM. Every training forward pass executes all blocks. The objective is

$$\mathcal{L} = \mathcal{L}_{\mathrm{task}} + \lambda_{\mathrm{KL}} D_{\mathrm{KL}}(p_{\mathrm{base}} \| p_{\mathrm{full}}) - \lambda_{\mathrm{ent}} \mathcal{H}(\alpha). \tag{7}$$

Here $\mathcal{L}_{\mathrm{task}}$ is the cross entropy for next token prediction. The distributions p_{base} and p_{full} are the outputs of the frozen teacher and the adapted model, respectively, and $\mathcal{H}(\alpha)$ is the mean routing entropy. The KL term penalizes departures from the teacher's outputs, and the entropy term discourages premature concentration of routing weights. Training omits no layers; the ReSkip policy is calibrated separately after adaptation.

3.3 ReSkip: selective execution using block boundary signals

ReSkip directly reuses the native routing weights that AttnRes computes for aggregation across blocks. Specifically, it uses the weight on the most recent source, $\alpha_{n-1\to n}(p)$ in Eq. 2, optionally combined with the RSM gain $g_n(p)$ in Eq. 4 as an additional criterion. Both signals come from the existing AR-Retrofit forward path and are available before the target block executes, so no additional skip controller needs to be trained. After offline calibration, ReSkip makes one decision for the current token at the target block entrance. When triggered, it omits a fixed subset of decoder layers within that block.

After adaptation, we freeze all model parameters and evaluate sensitivity on calibration sequences. These measurements guide the choice of the candidate block set $\mathcal{P}$ and the fixed layer offset set $\mathcal{S}_n \subseteq \{0, \ldots, L_{\mathrm{blk}} - 1\}$ for each block. We choose thresholds from empirical quantiles of the calibration signals (details in Appendix A2.1). The online rule is

$$z_n(p) = \mathbf{1}[n \in \mathcal{P}]\, \mathbf{1}[\alpha_{n-1\to n}(p) \ge \tau_n^{\alpha}]\, \mathbf{1}[g_n(p) < \tau_n^{g}]. \tag{8}$$

Here τ_n^{α} is a lower threshold on the weight of the most recent source, and τ_n^{g} is an upper threshold on the RSM gain. If the RSM condition is not used, we set $\tau_n^{g} = +\infty$. A single decision controls all layers in $\mathcal{S}_n$; no further scores are computed within the block, and the decision from the preceding token is not reused.

When $z_n(p) = 1$, layers at offsets in $\mathcal{S}_n$ are bypassed within block n. The skip branch retains the normalization, projections, RoPE, and cache writes needed to produce K/V for the current position. It omits the query projection, attention scoring and aggregation, output projection, and MLP. The hidden state remains unchanged at that layer, while the cache advances to the current position. Layers outside the selected subset execute normally. For LLM and VLM tasks, ReSkip applies only to decoding with a KV cache; prefill always follows the full path.

4 EXPERIMENTS

We evaluate whether AR-Retrofit can add adaptive depth while preserving pretrained capabilities, and whether ReSkip can accelerate inference without further training. The experiments cover language, multimodal understanding, and embodied manipulation. We measure quality and inference speed, examine how execution varies with the input, and compare methods with the same adaptation budget.

4.1 EXPERIMENTAL SETUP

Models and adaptation. We apply AR-Retrofit to four pretrained VLMs: Qwen3-VL-2B and 4B (Bai et al., 2025), InternVL3.5-2B (Wang et al., 2025), and SmolVLM2-2.2B (Marafioti et al., 2025). During AR-Retrofit adaptation, all pretrained backbone parameters remain frozen; only the added routers, bottleneck adapters, and residual strength modulation (RSM) modules are optimized. The scalar gate γ_n follows a deterministic linear schedule from zero to one, preserving the pretrained forward function at initialization and gradually introducing cross-depth aggregation. Unless otherwise stated, we use four layers per AttnRes block, a bottleneck dimension of $r = 256$, and 3,000 training steps. AR-Retrofit adds approximately 7.4M and 11.87M trainable parameters to Qwen3-VL-2B and 4B, corresponding to less than 0.4% and 0.3% of the respective backbones. Mean training loop time for 3,000 steps ranges from 12.6 to 33.8 minutes on a single H100 across the four backbones. Training costs, data, and Qwen3-VL block configurations are provided in Appendix A3.

Tasks and baselines. We evaluate seven text tasks in a zero-shot setting: LAMBADA (Paperno et al., 2016), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), ARC-Easy and ARC-Challenge (Clark et al., 2018), OBQA (Mihaylov et al., 2018), and WinoGrande (Sakaguchi et al., 2020). The six vision-language benchmarks are AI2D (Kembhavi et al., 2016), MMBench (Liu et al., 2024b), MMMU (Yue et al., 2024), MMStar (Chen et al., 2024), OCRBench (Liu et al., 2024c), and RealWorldQA (xAI, 2024). For vision-language-action (VLA) evaluation, we use the Spatial, Object, Goal, and LIBERO-10 suites of LIBERO (Liu et al., 2023). We attach an action head based on OpenVLA-OFT (Kim et al., 2025) to Qwen3-VL-2B and 4B and predict 8×7 action chunks with L_1 regression. During subsequent VLA policy training on the `libero_all` mixture, both Base and AR-Retrofit freeze the vision encoder and train the language backbone. We evaluate 500 episodes per suite, totaling 2,000 episodes per policy.

Table 1: VLM and LLM scores. Bold marks the better score within each pair; shaded rows show unweighted task averages.

Task	Qwen3-VL-2B		Qwen3-VL-4B		InternVL3.5-2B		SmolVLM2-2.2B	
	Base	ARR (ours)	Base	ARR (ours)	Base	ARR (ours)	Base	ARR (ours)
VLM tasks								
AI2D	75.65	**76.74**	81.80	**82.60**	75.91	**76.52**	66.35	**67.27**
MMBench	76.20	**77.95**	83.33	**85.14**	77.15	**78.46**	68.04	**69.01**
MMMU	42.33	**45.44**	50.44	**52.22**	49.33	**49.78**	35.78	**36.70**
MMStar	53.75	**55.77**	61.79	**62.32**	59.33	**60.23**	44.55	**46.76**
OCRBench	80.70	**84.27**	82.60	**85.47**	83.10	**83.30**	60.20	**61.23**
RealWorldQA	65.23	**66.19**	71.63	**72.51**	60.65	**61.57**	**58.43**	58.39
VLM Avg.	65.65	**67.73**	71.93	**73.37**	67.58	**68.31**	55.56	**56.56**
LLM tasks								
LAMBADA	52.49	**57.73**	60.94	**62.80**	57.09	**58.12**	65.67	**66.50**
HellaSwag	61.40	**63.16**	69.85	**71.44**	65.32	**65.97**	**67.68**	67.49
PIQA	73.94	**75.08**	77.09	**78.18**	73.88	**75.23**	74.27	**74.43**
ARC-E	72.26	**79.00**	78.62	**83.38**	75.25	**77.74**	**69.57**	69.53
ARC-C	47.18	**50.85**	55.72	**58.96**	48.72	**49.57**	44.11	**44.97**
OBQA	**38.40**	38.20	40.40	**41.60**	37.00	**37.80**	40.80	**41.07**
WinoGrande	62.27	**63.54**	66.77	**66.93**	60.62	**60.69**	65.43	**65.77**
LLM Avg.	58.28	**61.08**	64.20	**66.18**	59.70	**60.73**	61.08	**61.39**

We compare three configurations. **Base** uses the original pretrained backbone without AR-Retrofit; for VLA, it uses the same action head and LIBERO training protocol but no retrofit modules. **AR-Retrofit (ARR)** uses the adapted model with all blocks executed in full. **ReSkip** uses the same adapted weights with a fixed selective execution policy calibrated offline and requires no additional parameter training. Average scores (Avg.) are unweighted means over the tasks within each domain. Task scores and averages are rounded independently to two decimal places.

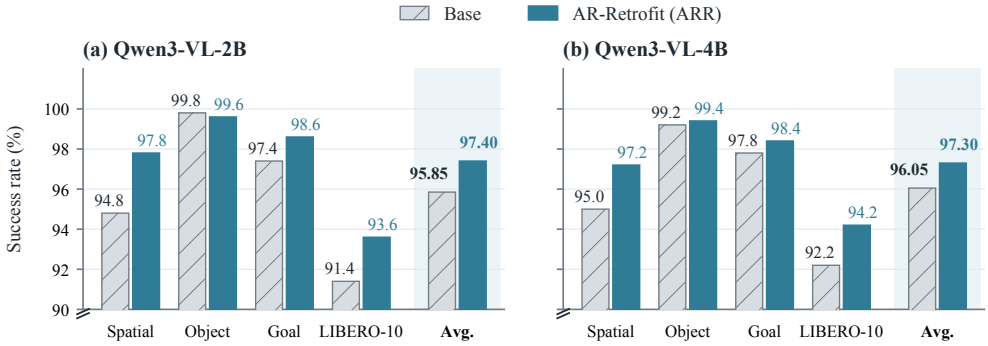


Figure 2: Success rates on four LIBERO suites. Gray hatched bars show Base. Blue bars show ARR.

4.2 Preserving and Improving Task Capabilities

Table 1 compares Base and ARR on LLM and VLM tasks. Across the seven text tasks, ARR improves the average LLM score on all four backbones, with gains of 2.80 and 1.98 points for Qwen3-VL-2B and 4B, respectively. The improvements span language modeling, commonsense reasoning, and question answering tasks.

On VLM benchmarks, Qwen3-VL-2B and 4B improve on all six benchmarks, with mean gains of 2.08 and 1.44 points. For Qwen3-VL-2B, the largest gains are 3.57 points on OCRBench and 3.11 points on MMMU. InternVL3.5-2B and SmolVLM2-2.2B also improve their average VLM scores by 0.73 and 1.00 points. ARR improves the average LLM and VLM scores on all four backbones.

The gains extend to closed-loop control (Figure 2). With the same action head and LIBERO training protocol, ARR reaches average success rates of 97.40% and 97.30% on Qwen3-VL-2B and 4B, exceeding Base by 1.55 and 1.25 percentage points, respectively.

Table 2: ReSkip inference speed and task quality in BF16 with batch size 1. Speed denotes decode throughput for LLM and VLM tasks and latency per action chunk for VLA. Bold marks the best speed in each row. Quality is the unweighted task average on a 0–100 scale, with Δ relative to Base or full ARR. Task scores and timing protocols are in Appendices A4 and A5.

Model	Task type	Speed			Quality		
		Base	ARR (ours)	ReSkip	ReSkip	Δ vs. Base	Δ vs. ARR
2B	LLM (tok/s ↑)	150.61	147.99	**152.94**	60.89	+2.61	−0.19
	VLM (tok/s ↑)	150.54	147.94	**163.80**	67.71	+2.06	−0.02
	VLA (ms ↓)	79.98	84.26	**79.92**	95.95	+0.10	−1.45
4B	LLM (tok/s ↑)	**97.04**	92.36	95.39	65.77	+1.57	−0.41
	VLM (tok/s ↑)	97.07	95.38	**101.44**	73.01	+1.08	−0.36
	VLA (ms ↓)	79.14	81.33	**78.59**	96.85	+0.80	−0.45

4.3 Quality and Speed with ReSkip

For ReSkip calibration, we freeze all model weights and construct candidate policies through offline sensitivity analysis and threshold calibration. We calibrate thresholds using the most recent source weight and optionally the RSM gain (Eq. 8). At most three candidate blocks can trigger selective execution per token. For LLM and VLM tasks, selective execution applies only during cached

decoding; for VLA, decisions are made per token during action prediction. Calibration results guide the choice of the main LLM/VLM operating points (Appendix A2). Timing precision and protocols are in Appendix A5.1.

Table 2 summarizes quality and inference speed across the three task domains. On VLM tasks, ReSkip stays within 0.4 points of ARR on average while reducing decode latency by 6.0% to 9.7%. It also improves throughput over Base by 4.5% to 8.8%, fully offsetting the routing and adaptation overhead. The speed gains are smaller on LLM tasks: relative to Base, 2B is 1.55% faster, whereas 4B is 1.70% slower. On VLA tasks, ReSkip reduces latency by 3.37% to 5.15% relative to ARR, with a decrease of 0.45 to 1.45 percentage points in success rate. Its latency is close to or slightly below Base. ReSkip selectively skips layer computations through offline calibration while preserving most of ARR's accuracy gains. Paired confidence intervals for LLM and VLM timing are in Appendix A5.2.

4.4 Mechanism Analysis

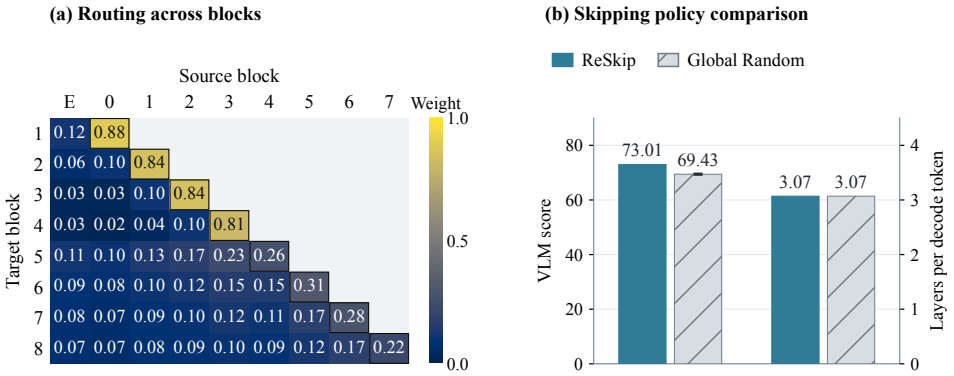


Figure 3: Routing and skipping on Qwen3-VL-4B. (a) Routing weights under full ARR, averaged over 28,124 answer tokens from 128 requests. Block IDs start at zero, and gray cells mark unavailable sources. (b) ReSkip and Global Random compared by average VLM score on the left axis and bypassed layers per decode token on the right. The Global Random error bar shows the sample standard deviation across three random seeds.

Figure 3 examines depth aggregation and the choice of layers to skip. In panel (a), the mean weight on the most recent source is 0.81 to 0.88 in blocks 1 to 4 and falls to 0.22 to 0.31 in blocks 5 to 8. Deeper blocks therefore assign more weight to earlier representations. Routing weights also vary across requests within the same block. For block 5, we average the total weight on earlier sources over each request's answer tokens. Across requests, the 10th and 90th percentiles are 0.65 and 0.74.

Panel (b) compares ReSkip with Global Random using the same ARR checkpoint. Both bypass an average of 3.07 decoder layers per decode token; Global Random samples from all 36 layers. ReSkip achieves an average VLM score 3.57 points higher, retaining more task performance at the same average bypass budget. Routing statistics and random control details are in Appendices A6 and A7.1.

4.5 Ablation Analysis

RSM strength. We replace 0.5 in Eq. 4 with s; $s = 0$ fixes the gain at one. All three nonzero settings improve the average VLM score (Table 3). The default $s = 0.50$ gains 0.54 points over $s = 0$; $s = 0.75$ is 0.17 points below the default. These results support RSM's adaptive control of residual correction strength across tokens and depths, with moderate modulation giving the best average score. Task scores and modulation ranges are in Appendix A7.2.

Table 3: RSM strength on Qwen3-VL-2B. Δ is relative to $s = 0$. Shading marks the default.

Strength s	VLM Avg.	Δ
0.00	67.19	—
0.25	67.45	+0.26
0.50	**67.73**	**+0.54**
0.75	67.56	+0.37

Comparison under the same adaptation budget. Table 4 compares adaptation methods using the same training data and 3,000 training steps, with approximately 12M trainable parameters per method. ARR achieves an average VLM score 1.42 points higher than LoRA and 1.09 points higher than DoRA. ReSkip retains a 1.06 point advantage over LoRA while increasing decode throughput by 4.38%. Training details and scores for individual tasks are in Appendix A8.

Table 4: Adaptation comparison on Qwen3-VL-4B-Instruct. Base is unadapted, and ReSkip reuses ARR weights. Timing is measured separately from Table 2, using the protocol in Appendix A8.2.

Method	Trainable params (M)	VLM Avg. ↑	Decode (tok/s) ↑
Base	0	71.93	88.06
LoRA	11.80	71.95	87.94
DoRA	11.98	72.28	87.74
FlexiDepth for VLMs	12.40	67.94	80.28
Block Adapter	11.80	59.40	85.42
ARR (ours)	11.87	**73.37**	86.40
ReSkip	11.87	73.01	**91.79**

ReSkip thresholds. Across the four coverage levels in Table 5, higher coverage increases bypassed computation and lowers average VLM scores for both model sizes. The main configurations keep the average score loss below 0.4 points while accelerating decoding (Table 2).

Table 5: ReSkip threshold ablation on six VLM tasks. Bypassed is the mean number of skipped layers per decode token divided by four. Δ is the score difference from full ARR, whose averages are 67.73 for 2B and 73.37 for 4B. Shading marks the main configurations.

Target coverage	2B			4B		
	Bypassed	Avg. ↑	Δ	Bypassed	Avg. ↑	Δ
0.10	0.69	67.84	$+0.11$	0.49	73.08	-0.29
0.25	0.81	67.72	-0.01	**0.77**	**73.01**	$\mathbf{-0.36}$
0.45	**0.96**	**67.71**	$\mathbf{-0.02}$	1.09	72.24	-1.13
0.65	1.02	65.66	-2.07	1.30	71.64	-1.73

Block size. On Qwen3-VL-2B, the setting with four layers per block achieves a VLM average of 67.73 after 3,000 steps, compared with 65.99 and 64.82 for two and one layer per block, using half and one quarter as many added parameters, respectively. Finer blocks improve with longer training but have higher prefill latency. These results support our default of four layers per block under the chosen training budget (Appendix A7.3).

5 Conclusion

We presented AR-Retrofit, which adds AttnRes aggregation to frozen pretrained decoders through residual corrections with few trainable parameters. It improves average language and multimodal performance and provides a stronger initialization for LIBERO policies. After offline calibration, ReSkip reuses its routing signals for selective execution without training a separate controller. On VLM tasks, it retains most gains and increases decode throughput over Base. Skip positions and thresholds require calibration for each model and modality, and quality depends on the adaptation data. Together, the methods share signals for representation aggregation and computation scheduling without full retraining. Future work will evaluate quality and efficiency on larger pretrained models.

AI USE STATEMENT

In this work, we used generative AI tools solely for language polishing and editing to aid the writing of the manuscript. We have not used generative AI tools for generating research ideas, designing or conducting experiments, analyzing or interpreting data, or producing scientific results. The remaining required disclosure tasks are not applicable to this work. Additionally, we did not use generative AI tools for tasks with recommended disclosure. We have reviewed all AI-assisted work. Specifically, all AI-assisted language edits were reviewed and verified by the authors for accuracy, clarity, and preservation of the original scientific meaning; any suggested changes were accepted only after author review. Details are described in the paper. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of AI.

REFERENCES

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Andrea Banino, Jan Balaguer, and Charles Blundell. PonderNet: Learning to ponder. In *8th ICML Workshop on Automated Machine Learning*, 2021. URL https://arxiv.org/abs/2107.05407.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7432–7439, 2020. doi: 10.1609/aaai.v34i05.6239. URL https://ojs.aaai.org/index.php/AAAI/article/view/6239.

Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, et al. Are we on the right way for evaluating large vision-language models? *Advances in Neural Information Processing Systems*, 37:27056–27087, 2024.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

James J DiCarlo, Davide Zoccolan, and Nicole C Rust. How does the brain solve visual object recognition? *Neuron*, 73(3):415–434, 2012.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3029–3051, 2023.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SJg7KhVKPH.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, 2024.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1380. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/3b01972cf31e6fa0fe29e4b8b5c2a0a1-Abstract-Conference.html.

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

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A. Roberts. The unreasonable ineffectiveness of the deeper layers. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/cbabc2f70de2dd09f491a8715ec3e80f-Abstract-Conference.html.

Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, et al. OpenThoughts: Data recipes for reasoning models. *arXiv preprint arXiv:2506.04178*, 2025.

Shwai He, Guoheng Sun, Zheyu Shen, and Ang Li. Uncovering the redundancy in transformers via a unified study of layer dropping. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=1I7PCbOPfe.

Ahmed Heakl, Martin Gubri, Salman Khan, Sangdoo Yun, and Seong Joon Oh. Dr.LLM: Dynamic layer routing in LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=3ow7tq0O3l.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pp. 2790–2799. PMLR, 2019.

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 *International Conference on Learning Representations*, 2022.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pp. 646–661. Springer, 2016.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Daniel Kahneman. *Thinking, fast and slow*. macmillan, 2011.

Kohitij Kar, Jonas Kubilius, Kailyn Schmidt, Elias B Issa, and James J DiCarlo. Evidence that recurrent circuits are critical to the ventral stream's execution of core object recognition behavior. *Nature neuroscience*, 22(6):974–983, 2019.

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*, pp. 235–251. Springer, 2016.

Tim C Kietzmann, Courtney J Spoerer, Lynn KA Sörensen, Radoslaw M Cichy, Olaf Hauk, and Nikolaus Kriegeskorte. Recurrence is required to capture the representational dynamics of the human visual system. *Proceedings of the National Academy of Sciences*, 116(43):21854–21863, 2019.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

Kimi Team, Guangyu Chen, Yu Zhang, Jianlin Su, Weixin Xu, Siyuan Pan, Yaoyu Wang, Yucheng Wang, Guanduo Chen, Bohong Yin, et al. Attention residuals. *arXiv preprint arXiv:2603.15031*, 2026.

Victor AF Lamme and Pieter R Roelfsema. The distinct modes of vision offered by feedforward and recurrent processing. *Trends in neurosciences*, 23(11):571–579, 2000.

Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Peiyuan Zhang, Yanwei Li, Ziwei Liu, et al. Llava-onevision: Easy visual task transfer. *arXiv preprint arXiv:2408.03326*, 2024a.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. NuminaMath. https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf, 2024b.

Xiang Lisa Li and Percy Liang. Prefix-Tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353/.

Ziyue Li, Yang Li, and Tianyi Zhou. Skip a layer or loop it? test-time depth adaptation of pretrained LLMs. *arXiv preprint arXiv:2507.07996*, 2025. doi: 10.48550/arXiv.2507.07996. URL https://arxiv.org/abs/2507.07996.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *Advances in Neural Information Processing Systems*, volume 35, 2022. doi: 10.52202/068431-0142. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/0cde695b83bd186c1fd456302888454c-Abstract-Conference.html.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32100–32121. PMLR, 2024a. URL https://proceedings.mlr.press/v235/liu24bn.html.

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, et al. Mmbench: Is your multi-modal model an all-around player? In *European conference on computer vision*, pp. 216–233. Springer, 2024b.

Yuliang Liu, Zhang Li, Mingxin Huang, Biao Yang, Wenwen Yu, Chunyuan Li, Xu-Cheng Yin, Cheng-Lin Liu, Lianwen Jin, and Xiang Bai. Ocrbench: on the hidden mystery of ocr in large multimodal models. *Science China Information Sciences*, 67(12):220102, 2024c.

Xuan Luo, Weizhi Wang, and Xifeng Yan. Adaptive layer-skipping in pre-trained LLMs. In *Second Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2503.23798.

Andrés Marafioti, Orr Zohar, Miquel Farré, Merve Noyan, Elie Bakouch, Pedro Cuenca, Cyril Zakka, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, Vaibhav Srivastav, Joshua Lochner, Hugo Larcher, Mathieu Morlon, Lewis Tunstall, Leandro von Werra, and Thomas Wolf. SmolVLM: Redefining small and efficient multimodal models. *arXiv preprint arXiv:2504.05299*, 2025. doi: 10.48550/arXiv.2504.05299. URL https://arxiv.org/abs/2504.05299.

Xin Men, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20192–20204, 2025.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, 2018. doi: 10.18653/v1/D18-1260. URL https://aclanthology.org/D18-1260/.

OpenAI. Learning to reason with LLMs. https://openai.com/index/learning-to-reason-with-llms/, 2024.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 1525–1534, 2016.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8732–8740, 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/6399.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472, 2022.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Courtney J Spoerer, Tim C Kietzmann, Johannes Mehrer, Ian Charest, and Nikolaus Kriegeskorte. Recurrent neural networks can explain flexible trading of speed and accuracy in biological vision. *PLoS computational biology*, 16(10):e1008215, 2020.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, et al. InternVL3.5: Advancing open-source multimodal models in versatility, reasoning, and efficiency. *arXiv preprint arXiv:2508.18265*, 2025. doi: 10.48550/arXiv.2508.18265. URL https://arxiv.org/abs/2508.18265.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

xAI. RealWorldQA. https://huggingface.co/datasets/xai-org/RealworldQA, 2024.

Zhenda Xie, Yixuan Wei, Huanqi Cao, Chenggang Zhao, Chengqi Deng, Jiashi Li, Damai Dai, Huazuo Gao, Jiang Chang, Kuai Yu, Liang Zhao, Shangyan Zhou, Zhean Xu, Zhengyan Zhang, Wangding Zeng, Shengding Hu, Yuqing Wang, Jingyang Yuan, Lean Wang, and Wenfeng Liang. mHC: Manifold-constrained hyper-connections. *arXiv preprint arXiv:2512.24880*, 2026. doi: 10.48550/arXiv.2512.24880. URL https://arxiv.org/abs/2512.24880v2. Version 2.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9556–9567, 2024.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. Hyper-connections. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/f1e8ff97057e0c3f6bcb7018c78d4df1-Abstract-Conference.html.

A1 TECHNICAL DETAILS AND THEORETICAL ANALYSIS OF AR-RETROFIT

This appendix supplements Section 3.2 with architectural details, a proof of equivalence at initialization, and an analysis of gradient propagation.

A1.1 EQUIVALENCE AT INITIALIZATION

Proposition. When $\gamma_n(0) = 0$, the retrofitted model has the same forward function as the pretrained model at initialization.

Proof. The embedding stream h_0 and all Block_n transformations are unchanged, so the first block produces the same output as the pretrained model. Assume that the output of block $n-1$ is also unchanged. With $\gamma_n(0) = 0$, Eq. 4 gives

$$x_n = h_{n-1} + 0 \cdot g_n A_n(\delta_n) = h_{n-1}.$$

Thus $h_n = \mathrm{Block}_n(h_{n-1})$ is also identical to the pretrained model's output. By induction, all block outputs are identical. This result holds in exact arithmetic and does not require the router or adapter to approximate the identity map.

RSM initialization. We initialize w_n^{RSM} and b_n^{RSM} to zero, giving $g_n = 1$. For any fixed γ_n, this preserves the forward function of AR-Retrofit without RSM.

A1.2 ARCHITECTURAL DETAILS

Source bias. The source bias b_i contributes a score $w_n^\top b_i/\sqrt{d}$ that depends on source depth but not content. When $b_i = 0$, treating $w_n/\sqrt{d}$ as a reparameterized query recovers the scoring form in Eq. 1.

Bottleneck adapter. The adapter has no biases: $A_n(u) = W_n^{\mathrm{up}}\sigma(W_n^{\mathrm{down}}u)$, where $W_n^{\mathrm{down}} \in \mathbb{R}^{r\times d}$ and $W_n^{\mathrm{up}} \in \mathbb{R}^{d\times r}$. Its output lies in the column space of W_n^{up}, whose dimension is at most r, and $A_n(0) = 0$.

Training objective weights. The mean routing entropy in Eq. 7 is

$$\mathcal{H}(\alpha) = \mathbb{E}_{n,p}\left[-\sum_{i<n}\alpha_{i\to n}(p)\log\alpha_{i\to n}(p)\right].$$

The coefficients λ_{KL} and λ_{ent} control the weights of the KL divergence and entropy regularization terms, respectively.

A1.3 GRADIENT PROPAGATION AND INITIALIZATION STRATEGY

Adapter initialization. Both adapter projections use small random initializations. The up projection is nonzero to avoid blocking gradients, while $\gamma_n(0) = 0$ guarantees equivalence to the pretrained model at initialization.

Gradient propagation. Holding the predecessor state fixed and RSM at its identity initialization, the local Jacobian of the corrected input with respect to the routed state is

$$\frac{\partial x_n}{\partial r_n} = \gamma_n W_n^{\mathrm{up}}\,\mathrm{diag}\big[\sigma'\big(W_n^{\mathrm{down}}\delta_n\big)\big]\,W_n^{\mathrm{down}}.$$

If the up projection is zero, this Jacobian is zero. Neither the router nor the down projection can then receive gradients through this path when the gate first opens. A nonzero up projection removes this structural obstruction, while $\gamma_n(0) = 0$ still guarantees equivalence at initialization.

Training schedule. Training uses a deterministic schedule $\gamma_n(t) = s(t)$, with $0 \le s(t) \le 1$, $s(0) = 0$, and $s(T) = 1$. Here t is the training step and T is the end of the gate ramp. The gates γ_n are not optimized by gradient descent.

Continuity of the injection magnitude. For fixed historical states and adapter parameters,

$$\|x_n - h_{n-1}\| \le 1.5\gamma_n\|A_n(\delta_n)\|.$$

The injection magnitude therefore tends continuously to zero as $\gamma_n \to 0$.

A2 RESKIP CALIBRATION

ReSkip uses offline sensitivity analysis and threshold calibration to construct candidate policies. A candidate block triggers selective execution when its most recent source weight $\alpha_{n-1\to n}$ meets the lower threshold τ^{α} and, if used, its RSM gain satisfies $g_n < \tau^{g}$. At most three candidate blocks can trigger for each token.

A2.1 SENSITIVITY ANALYSIS

For the 2B and 4B LLM/VLM policies, we use the same 32 instruction records from LLaVA-OneVision, UltraChat, and NuminaMath: 16 for threshold fitting and 16 for diagnostics.

For LLM/VLM, we measure sensitivity on calibration sequences using $D_{\mathrm{KL}}(p_{\mathrm{full}} \| p^{-(n,\ell)})$ and whether the highest probability token changes. Here $p^{-(n,\ell)}$ is the output distribution when only the layer at offset ℓ in block n is forcibly omitted. For a layer subset, the risk is the maximum KL divergence over its individual layer omissions, plus 0.25 if any omission changes the highest probability token. These measurements guide the choice of the candidate block set $\mathcal{P}$ and the fixed layer offset set $\mathcal{S}_n \subseteq \{0, \ldots, L_{\mathrm{blk}} - 1\}$ for each block.

We choose thresholds from empirical quantiles of the calibration signals. Conditional risk is the mean subset risk over positions where the rule triggers. When using only the routing weight, we prioritize matching the target coverage and use conditional risk to break ties. When also using RSM, we minimize the sum of conditional risk and deviation from target coverage within a predefined coverage interval. Once the final policy is selected (Appendix A2.3), its thresholds, $\mathcal{P}$, and $\mathcal{S}_n$ remain fixed.

For VLA, we collect 16 trajectories per model size with all layers executed, using fresh simulator resets of the first two tasks in each LIBERO suite. We use eight trajectories to fit thresholds and eight to select the operating point based on action drift and changes in gripper commands. Calibration initial states do not exactly match any official evaluation initial state for these tasks. We freeze each policy before official LIBERO evaluation, without using evaluation success rates for selection.

A2.2 LAYER EXECUTION RULE

Let $h_{n,\ell}(p)$ denote the state between layers within a block, with $h_{n,0}(p) = x_n(p)$ and $h_{n,L_{\mathrm{blk}}}(p) = h_n(p)$. Omitting cache arguments, the execution rule is

$$h_{n,\ell+1}(p) = \begin{cases} h_{n,\ell}(p), & z_n(p) = 1,\ \ell \in \mathcal{S}_n, \\ \mathrm{Layer}_{n,\ell}(h_{n,\ell})(p), & \text{otherwise.} \end{cases} \tag{A2.1}$$

Here $\mathrm{Layer}_{n,\ell}$ is the original decoder layer, and $z_n(p)$ is the decision in Eq. 8.

A2.3 MAIN EXPERIMENT CONFIGURATIONS

For each model size, calibration results guide the selection of candidate policies and a single LLM/VLM operating point. The selected policy and thresholds are fixed before benchmark evaluation and remain unchanged across all six reported VLM benchmarks and subsequent timing measurements.

Table A2-1 lists the policies used in the main experiments. Each model size uses a separate policy for VLA. Within each setting, quality evaluation and timing use the same checkpoint. Block IDs and layer offsets start from zero.

Table A2-1: ReSkip policies for the main experiments. Routing thresholds follow the order of candidate blocks and are shown in units of 10^{-3}.

Scale	Domain	Blocks: layer offsets	τ^{α}	RSM
2B	LLM/VLM	$\mathrm{b1}:\{1,3\};\ \mathrm{b2}:\{1\};\ \mathrm{b3}:\{0,1\}$	$(887, 836, 406)$	b1: $g < 523 \times 10^{-3}$
4B	LLM/VLM	$\mathrm{b1}:\{1,2,3\};\ \mathrm{b3}:\{0,1\};\ \mathrm{b4}:\{1\}$	$(914, 855, 824)$	None
2B	VLA	$\mathrm{b2}:\{1\};\ \mathrm{b3}:\{0,1\}$	$(996, 473)$	None
4B	VLA	$\mathrm{b1}:\{1,2,3\};\ \mathrm{b3}:\{0,1\};\ \mathrm{b4}:\{1\}$	$(1000, 996, 996)$	None

For language and vision tasks, selective execution is enabled only during decoding with a KV cache; prefill executes every layer in full. For VLA, decisions are made for each token during every forward pass that predicts an 8×7 action chunk.

A3 SUPPLEMENTARY EXPERIMENTAL SETUP

Table A3-1 lists the block configurations for the Qwen3-VL models.

Table A3-1: AR-Retrofit configurations for Qwen3-VL. D is the number of backbone layers, d the hidden dimension, N the number of blocks, and r the bottleneck dimension. Added parameter counts are approximate.

Backbone	D	d	N	Layers/block	r	Added (M)
Qwen3-VL-2B	28	2048	7	4	256	7.4
Qwen3-VL-4B	36	2560	9	4	256	11.87

The training mixture combines LLaVA-OneVision-Data (Li et al., 2024a), UltraChat (Ding et al., 2023), NuminaMath (Li et al., 2024b), and OpenThoughts (Guha et al., 2025) in a 6:2:1:1 ratio. Table A3-2 gives the settings for AR-Retrofit adaptation with a frozen backbone. InternVL3.5-2B and SmolVLM2-2.2B use the same settings as Qwen3-VL-2B and 4B. Subsequent VLA policy training follows the separate protocol in Section 4.1.

Table A3-2: AR-Retrofit adaptation hyperparameters shared by all four backbones.

Setting	Value
Optimizer	AdamW with $\beta_1 = 0.9$ and $\beta_2 = 0.95$
Peak learning rate	1×10^{-3}
Learning rate warmup	Linear, 100 steps
Learning rate decay	Cosine to 1×10^{-4}
Weight decay	0
Batch size	1
Effective batch size	1
Gradient accumulation	None
Training steps	3,000
Gate ramp T	900 steps, γ increases linearly from 0 to 1
λ_{KL}	1.0
λ_{ent}	0.02
KL temperature	1.0
Gradient clipping	1.0
Precision	BF16
Maximum sequence length	1,536 tokens
RSM scale	0.5

Table A3-3 reports the training loop time to reach step 3,000 and the corresponding H100 GPU hours for each backbone.

Table A3-3: Mean ARR adaptation cost for 3,000 steps on a single H100.

Backbone	Time (min)	H100 hours
Qwen3-VL-2B	12.6	0.209
Qwen3-VL-4B	16.2	0.270
InternVL3.5-2B	19.4	0.323
SmolVLM2-2.2B	33.8	0.564

For the language benchmarks in Section 4.1, we use accuracy for LAMBADA and WinoGrande, and `acc_norm` for the other tasks. Vision and language metrics are rescaled to 0–100 before computing the unweighted mean. The VLA action head is based on OpenVLA-OFT (Kim et al., 2025).

A4 ReSkip results for individual tasks

Table A4-1 reports detailed ReSkip scores. Corresponding Base and AR-Retrofit results are in Section 4.2.

Table A4-1: ReSkip scores for individual tasks on a 0–100 scale.

Domain	Task	2B ReSkip	4B ReSkip
LLM	LAMBADA	58.04	62.74
LLM	HellaSwag	62.67	69.65
LLM	PIQA	74.58	77.82
LLM	ARC-Easy	78.66	82.49
LLM	ARC-Challenge	50.09	58.17
LLM	OpenBookQA	38.80	42.20
LLM	WinoGrande	63.38	67.32
LLM	**Avg.**	**60.89**	**65.77**
VLM	AI2D	76.91	82.80
VLM	MMBench	78.69	84.02
VLM	MMMU	45.89	51.22
VLM	MMStar	56.05	62.37
VLM	OCRBench	82.20	84.80
VLM	RealWorldQA	66.54	72.81
VLM	**Avg.**	**67.71**	**73.01**
VLA	LIBERO-Spatial	95.60	97.20
VLA	LIBERO-Object	98.40	99.80
VLA	LIBERO-Goal	97.40	96.80
VLA	LIBERO-10	92.40	93.60
VLA	**Avg.**	**95.95**	**96.85**

A5 Supplementary inference measurements

A5.1 Timing configuration

Language timing uses WikiText-2 (Merity et al., 2016) samples with 512 input tokens and 128 generated tokens. Vision and language timing uses RealWorldQA samples with images resized to 448×448, 512 input tokens including 196 visual tokens, and 64 generated tokens. VLA timing uses LIBERO observations with 224×224 images, 256 input tokens, and an 8×7 output action chunk.

All measurements use an H100 GPU, BF16 precision, and batch size 1. AR-Retrofit and ReSkip share the same operator optimizations. We repeat each request three times, take the median within each request, and then take the median across requests.

A5.2 Statistical intervals

Table A5-2 reports percentage speed changes from paired timing measurements, calculated as $100(r - 1)$ from each speed ratio r. Positive values indicate that ReSkip is faster. We estimate the intervals by bootstrap with 10,000 resamples.

Table A5-1: LLM and VLM latency by stage (ms).

Scale	Domain	Method	Prefill	Decode	Total
2B	LLM	Base	20.16	843.26	863.40
2B	LLM	ARR (ours)	24.48	858.15	882.55
2B	LLM	ReSkip	24.56	830.37	855.53
2B	VLM	Base	35.16	418.49	453.61
2B	VLM	ARR (ours)	39.44	425.84	465.28
2B	VLM	ReSkip	39.42	384.62	425.11
4B	LLM	Base	26.00	1308.77	1335.11
4B	LLM	ARR (ours)	31.60	1375.05	1496.78
4B	LLM	ReSkip	31.88	1331.35	1363.50
4B	VLM	Base	40.66	649.04	689.94
4B	VLM	ARR (ours)	46.41	660.51	707.28
4B	VLM	ReSkip	46.39	621.04	667.41

Table A5-2: ReSkip speed changes relative to Base (%), with 95% confidence intervals.

Scale	Domain	Decode [95% CI]	Total [95% CI]
2B	LLM	2.46 [1.29, 3.83]	1.84 [0.71, 3.17]
2B	VLM	8.83 [7.82, 9.74]	6.93 [5.99, 7.77]
4B	LLM	-1.69 [-2.00, -1.37]	-2.09 [-2.41, -1.77]
4B	VLM	4.90 [4.00, 5.91]	3.65 [2.81, 4.60]

A6 Routing statistics of AR-Retrofit

Block IDs in this appendix start from zero.

For each request and token type, we sum the routing weights on all sources except the most recent one, then average over tokens of that type. Visual, prompt, and answer tokens have 91, 128, and 128 nonempty requests, with 21,962, 12,782, and 28,124 tokens, respectively. Requests without visual tokens are excluded from visual statistics.

Figure A6-1 and Table A6-1 summarize these distributions.

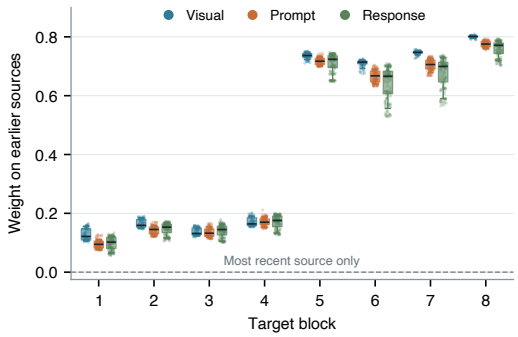


Figure A6-1: Routing variation in 4B AR-Retrofit. Points show request means by token type. Boxes show medians and interquartile ranges, with whiskers at the 10th and 90th percentiles.

Table A6-1: Total routing weight on earlier sources. Token means weight requests by token count. Other statistics weight nonempty requests equally. SD denotes standard deviation.

Target	Type	Token mean	Req. mean	SD	P10	Q25	Median	Q75	P90
1	Visual	0.14	0.13	0.02	0.11	0.11	0.12	0.15	0.16
1	Prompt	0.11	0.09	0.01	0.08	0.08	0.10	0.11	0.11
1	Answer	0.12	0.10	0.02	0.07	0.08	0.10	0.12	0.12
2	Visual	0.17	0.17	0.01	0.15	0.15	0.16	0.18	0.19
2	Prompt	0.15	0.14	0.01	0.12	0.13	0.15	0.15	0.16
2	Answer	0.16	0.15	0.02	0.12	0.13	0.15	0.16	0.17
3	Visual	0.14	0.14	0.01	0.13	0.13	0.13	0.15	0.16
3	Prompt	0.15	0.14	0.01	0.12	0.12	0.13	0.15	0.16
3	Answer	0.16	0.14	0.02	0.11	0.13	0.15	0.16	0.16
4	Visual	0.18	0.17	0.01	0.16	0.16	0.16	0.19	0.19
4	Prompt	0.18	0.17	0.01	0.15	0.16	0.17	0.18	0.19
4	Answer	0.19	0.17	0.02	0.13	0.15	0.18	0.19	0.20
5	Visual	0.74	0.74	0.01	0.72	0.73	0.74	0.74	0.75
5	Prompt	0.73	0.72	0.01	0.70	0.71	0.72	0.73	0.73
5	Answer	0.74	0.71	0.03	0.65	0.70	0.72	0.74	0.74
6	Visual	0.71	0.71	0.01	0.69	0.71	0.71	0.72	0.72
6	Prompt	0.68	0.67	0.02	0.64	0.65	0.67	0.68	0.69
6	Answer	0.69	0.64	0.06	0.55	0.61	0.67	0.68	0.70
7	Visual	0.75	0.75	0.01	0.74	0.75	0.75	0.75	0.75
7	Prompt	0.72	0.70	0.02	0.68	0.69	0.71	0.72	0.73
7	Answer	0.72	0.68	0.05	0.59	0.65	0.70	0.71	0.73
8	Visual	0.80	0.80	< 0.01	0.80	0.80	0.80	0.80	0.80
8	Prompt	0.78	0.78	0.01	0.76	0.77	0.78	0.78	0.79
8	Answer	0.78	0.76	0.03	0.72	0.74	0.77	0.78	0.79

The percentile intervals describe variation across requests, not confidence intervals or trajectories over tokens. The zero line represents using only the most recent source, not a measured Base weight. These statistics do not isolate the contribution of RSM.

A7 Supplementary ablation results

A7.1 Random skipping control

Both policies in Figure 3b share a Qwen3-VL-4B AR-Retrofit checkpoint and 11,582 requests across six VLM tasks. Global Random samples from all 36 layers. Its score is reported as the mean and sample standard deviation over three preset seeds. The 3.57-point difference is computed before rounding. Budgets match average bypassed layers per decode token. Generated lengths differ, so total computation and latency are not matched.

A7.2 RSM strength and modulation ranges

Table A7-1 gives the task scores for the strength settings in Table 3.

Table A7-1: RSM strength ablation on Qwen3-VL-2B. Shading marks the default strength.

Strength s	AI2D	MMBench	MMMU	MMStar	OCRBench	RealWorldQA	Avg.
0.00	76.52	77.95	45.11	54.92	83.80	64.84	67.19
0.25	76.59	77.87	45.11	55.81	83.70	65.62	67.45
0.50	76.74	77.95	45.44	55.77	84.27	66.19	**67.73**
0.75	76.39	77.27	45.56	55.93	84.20	66.01	67.56

Table A7-2 reports the observed modulation factor ranges for the RSM strengths in Table 3.

Table A7-2: Observed RSM gain ranges and the mean of the standard deviations within individual blocks.

Strength s	Observed range	Mean block SD
0.00	1.00 to 1.00	0.00
0.25	0.75 to 1.25	0.12
0.50	0.50 to 1.50	0.16
0.75	0.25 to 1.69	0.11

At $s = 0.75$, the observed maximum of 1.69 is below the theoretical upper bound of 1.75. Mean block SD is the average of the standard deviations of the modulation factors within individual blocks. It does not measure variation across repeated training runs.

A7.3 Block size and training budget

Table A7-3 varies block size and training budget on Qwen3-VL-2B.

Table A7-3: Average scores over six VLM tasks across training budgets. Bold marks the best evaluated score per configuration.

Layers/block	Blocks	Params (M)	3,000 steps	6,000 steps	9,000 steps	Best step
1	28	29.53	64.82	66.23	**68.46**	9,000
2	14	14.77	65.99	**67.88**	67.17	6,000
4	7	7.39	**67.73**	—	—	3,000

All runs start independently from the same pretrained backbone and use identical encoded inputs at equal budgets. Shorter runs use prefixes of longer runs. For each configuration, we select the best evaluated result by its average score over six tasks.

After 3,000 steps, mean prefill latency is 66.25, 56.08, and 50.43 ms for one, two, and four layers per block, respectively. With four layers per block in the numerator, geometric mean latency ratios across paired requests are 79.38% (95% CI: 79.27% to 79.50%) for one layer per block and 92.30% (95% CI: 92.21% to 92.40%) for two.

A8 Comparisons under a limited adaptation budget

This section gives the training and evaluation protocols for Table 4 in the main text. All methods use Qwen3-VL-4B-Instruct. Speed is measured on the same H100 with BF16 precision and batch size 1.

A8.1 Training protocols and results for individual tasks

All adaptation methods use the same 3,000 encoded training inputs, batch size 1, a maximum sequence length of 1,536, and an image limit of 802,816 pixels. We evaluate every adapted model at step 3,000. Base is not trained, and ReSkip reuses AR-Retrofit weights.

LoRA and DoRA use SFT alone. Their updates are merged into the backbone weights before evaluation and timing. Block Adapter is trained independently at the same positions as AR-Retrofit, without routing across blocks. It is not implemented by disabling routing in AR-Retrofit at inference.

FlexiDepth uses a fixed reduction ratio of 128 and 12,395,880 trainable parameters. It retains routing and FFN/adapter branches in the last 18 layers. Starting from Base, it trains for 3,000 steps with an alignment objective and penalties for routing and skipping, without additional annealing. The VLM adaptation preserves the attention and KV behavior of the released implementation. For timing, branch scheduling for single tokens is moved into a CUDA conditional graph, while skipping is still executed. These results describe the adaptation under this budget, not the performance ceiling of the full training recipe.

Table A8-1: VLM scores for the four comparison methods on a 0–100 scale. Evaluation uses the full task splits, including MMBench English dev static and MMMU validation. Avg. is the unweighted mean over six tasks.

Method	AI2D	MMBench	MMMU	MMStar	OCRBench	RealWorldQA	Avg.
LoRA	80.41	83.68	51.22	61.23	83.10	72.03	71.95
DoRA	81.00	83.68	52.78	60.65	83.40	72.16	72.28
FlexiDepth for VLMs	78.50	78.35	47.00	57.56	79.70	66.54	67.94
Block Adapter	62.76	66.41	37.11	51.04	76.60	62.48	59.40

Average scores are computed before rounding. Quality scores for Base, AR-Retrofit, and ReSkip come from Tables 1 and 2; all speed measurements use the separate protocol below.

A8.2 Matched timing protocol

All seven models reside simultaneously in the same process on one H100, with the SM clock fixed at 1,590 MHz, BF16 precision, and batch size 1. We use 12 fixed RealWorldQA requests, each with a 448×448 image, 512 input tokens, and 64 output tokens. After warmup, we rotate through models for each request and repeat each request three times. We take the median within each request and then across requests.

Throughput is $63/T_{\text{decode}}$, with time measured in seconds. Only the 63 cached decode steps after the first token are timed. We exclude prefill, input processing and transfer, cache resets, and graph construction. Each model passes equivalence checks for generated sequences and execution routes on two sets of real requests. These measurements are separate from Table 2 and Appendix A5 and do not reuse their paired intervals. The 6.24% throughput gain over AR-Retrofit is calculated from the displayed values in Table 4, as $91.79/86.40 - 1$.

A9 Execution depth

This appendix reports execution depth for the frozen 4B ReSkip policy.

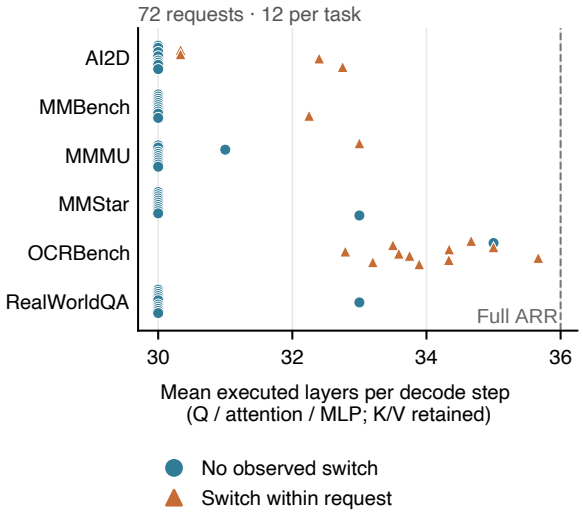


Figure A9-1: Mean executed depth during decoding for 4B ReSkip on 12 preselected requests per task. Key and value computations are retained. Triangles mark requests with changes in layer execution, while circles mark those with no observed change.

A9.1 EXECUTION DEPTH AND CHANGES WITHIN REQUESTS

We use 12 predefined requests per task, selected at evenly spaced positions in the original order without filtering by answers or skipped computation. Generated tokens and execution counts agree across three runs, so we count them only once.

The 4B model has 36 layers. Each trigger of candidate blocks b1, b3, and b4 bypasses Q, attention, and MLP computation in 3, 2, and 1 layers, respectively, while K/V still updates. For request i with T_i decode steps and trigger counts C_{i1}, C_{i3}, and C_{i4},

$$S_i = 3C_{i1} + 2C_{i3} + C_{i4}, \qquad \bar{D}_i = 36 - \frac{S_i}{T_i}. \tag{A9.1}$$

Here S_i counts omitted layer computations across tokens, and $\bar{D}_i$ is the mean executed depth, excluding full prefill. Because K/V computation is retained, this layer count does not translate proportionally into FLOPs or latency. We record an execution change within a request if any candidate satisfies $0 < C_{ib} < T_i$. This identifies whether a change occurs, not its timing or frequency.

Table A9-1: Execution depth and changes within 72 sampled requests. Depth ranges span the means of individual requests. Only requests with multiple decode steps are eligible for the change count.

Task	Requests	Decode steps	Depth range	Changed / eligible
AI2D	12	23	30.00–32.75	4 / 4
MMBench	12	17	30.00–32.25	1 / 3
MMMU	12	17	30.00–33.00	1 / 2
MMStar	12	12	30.00–33.00	0 / 0
OCRBench	12	307	32.79–35.67	11 / 11
RealWorldQA	12	13	30.00–33.00	0 / 1

Table A9-2: Candidate block trigger counts for the same requests. Each trigger omits the candidate's entire fixed layer subset. The last column counts omitted layer computations across tokens.

Task	Decode steps	b1	b3	b4	Omitted layers
AI2D	23	18	22	15	113
MMBench	17	15	16	16	93
MMMU	17	14	15	14	86
MMStar	12	11	12	12	69
OCRBench	307	77	168	71	638
RealWorldQA	13	12	13	13	75
Total	389	147	246	141	1,074

The subset contains 389 decode steps, giving a mean depth weighted by step count of $36 - 1{,}074/389 \approx 33.24$ layers. The full evaluation contains 49,372 decode steps and 151,481 omitted layer computations across tokens, giving a mean depth of $36 - 151{,}481/49{,}372 \approx 32.93$ layers.

Of the 21 requests with multiple decode steps, 17 change a candidate block's execution state during generation. Figure A9-1 retains all 72 points. Triangles mark requests with changes, and circles mark no observed change, including requests with only one step. Vertical offsets only prevent overlap. The dashed line at 36 layers denotes AR-Retrofit without ReSkip, labeled Full ARR.