ARCHITECTURAL SAMPLING: TEST-TIME SCALING VIA COMPUTATIONAL DIVERSITY IN FROZEN VISION-LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time scaling often seeks better answers by sampling multiple responses from a frozen model, yet conventional temperature sampling generates every candidate along the same fixed computation path. We introduce architectural sampling, a training-free method that generates candidates through distinct forward computations by reusing selected blocks of decoder layers. Varying the block location and repetition count introduces computational diversity without updating model weights or adding auxiliary parameters. Across five Qwen checkpoints and twelve multimodal benchmarks, architectural sampling improves pass@9 over standard-path temperature sampling by 6.58 percentage points on average at the same nine-candidate budget. Reusing early layers yields the strongest gains, and the improvement in candidate coverage persists even under greedy decoding. The resulting candidates show lower lexical overlap and improve accuracy when used as rollouts for label-free test-time reinforcement learning. These findings extend the benefits of our architectural sampling beyond candidate coverage, demonstrating more effective learning from a model's own outputs.

"No man ever steps in the same river twice, for it is not the same river and he is not the same man."
— Heraclitus

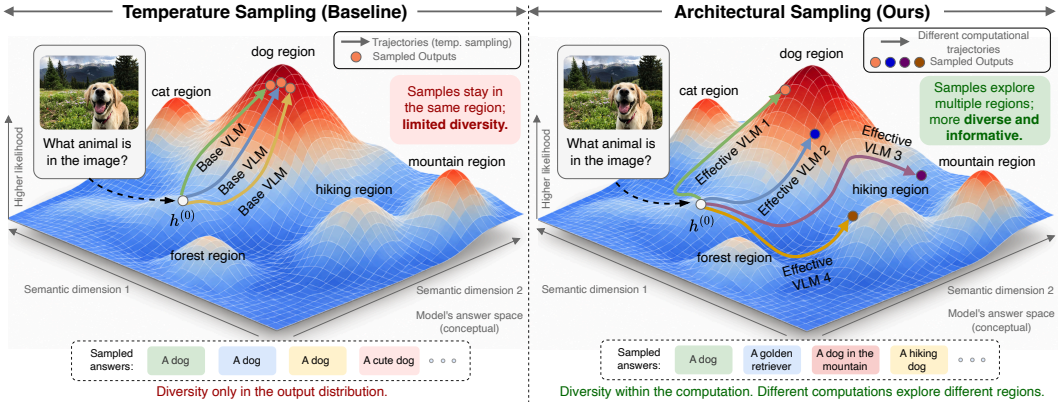


Figure 1: **Two Sources of Candidate Diversity.** Temperature sampling (left) draws several outputs from the same standard computation path, starting from the initial multimodal representation $h^{(0)}$. Architectural sampling (right) changes the computation path from the same $h^{(0)}$ before producing one candidate from each path. The background is a conceptual illustration of the answer space.

1 INTRODUCTION

Open-weight vision-language models (VLMs) are increasingly used exactly as they are shipped. A practitioner downloads a frozen checkpoint like Qwen-VL or Gemma and queries it directly, with no fine-tuning and no infrastructure beyond a forward pass (Bai et al., 2025a; Zhu et al., 2025).

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.

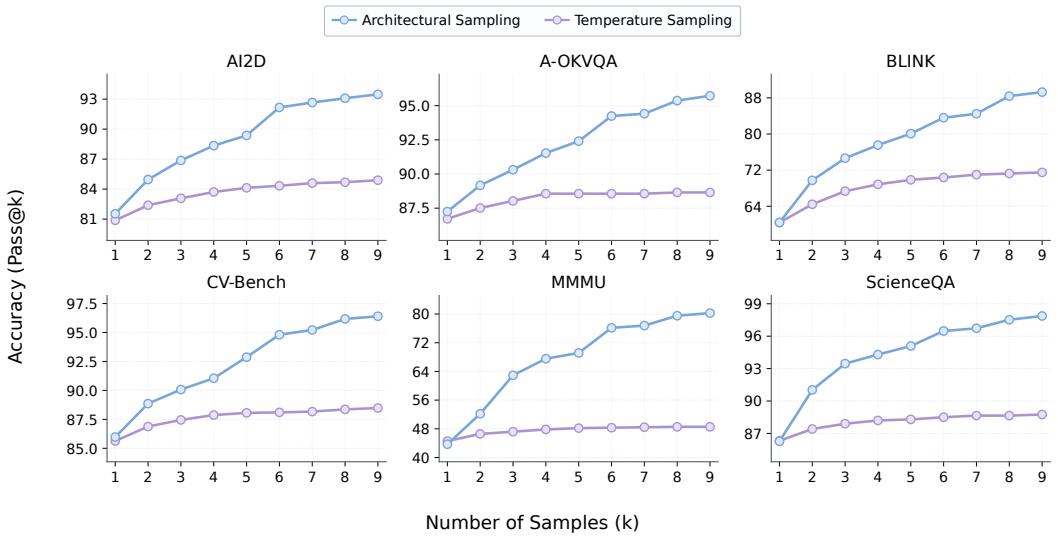


Figure 2: **Architectural sampling scales more favorably with candidate budget.** pass@k of Qwen3-VL-8B across six multimodal benchmarks as the candidate budget grows. Architectural sampling continues to improve candidate coverage as the pool grows, while temperature sampling plateaus earlier.

In this regime the only way to extract more from a model is to spend more resources *at inference*. This constraint has motivated a rapidly growing body of work on test-time scaling, in which a model generates multiple candidate answers and an aggregation, verification, or selection procedure distills them into a stronger final solution (Snell et al., 2025; Kaya et al., 2026; Singh et al., 2026).

The promise of these methods rests on a single load-bearing assumption: that a VLM can produce a *diverse* set of candidates that collectively provide a broader coverage of the solution space for downstream tasks. Yet this assumption conceals a fundamental question that the literature has largely left implicit: *where does candidate diversity actually come from?*

The question matters because candidate-based test-time scaling is fundamentally bounded by the candidates it is given. Generating k answers is useful only when those answers provide meaningful coverage of the solution space. If all k samples occupy essentially the same region of this space, increasing k provides little additional value. The dominant way to create this diversity is remarkably simple: *temperature sampling* (Holtzman et al., 2020). However, temperature operates only at the final stage of the generation process. Every candidate is routed through the same layers with the same parameters and the same intermediate representations, and only the final draw from the resulting distribution differs. Diversity, in other words, is injected after all the interesting computation has already happened. Figure 1 illustrates this contrast: standard-path temperature sampling varies the output draw after a common computation, whereas architectural sampling varies the computation that precedes it.

This raises a natural follow-up: *what if diversity need not come from changing the output distribution, but from changing the computation that produces it? Can a frozen model, without changing a single parameter, reveal new capabilities simply by exploring different ways of computing?*

We answer this question with *architectural sampling*, illustrated in Figure 3. For each candidate, we select a short contiguous block of decoder layers, apply it a chosen number of times, and then resume the original forward pass. Changing the block location and repetition count yields several computation paths from one checkpoint. The weights remain frozen throughout, and the method adds no auxiliary parameters. Each path can be decoded greedily or with temperature sampling, allowing architectural and token-level variation to be combined.

This introduces a new axis for test-time scaling centered around architecture, in which candidate diversity arises from varying the computation performed by a frozen model. This raises a natural question: *does architectural sampling reliably beat temperature sampling, and where do its gains come from?*

We observe consistent benefits across the VLM backbones and benchmarks evaluated. Across five Qwen checkpoints and twelve multimodal benchmarks, architectural sampling consistently improves pass@9 over temperature sampling, with an average gain of approximately 6.58%. This advantage is not confined to a single budget: Figure 2 traces pass@k for both methods as k grows, showing that architectural sampling scales more favorably and dominates temperature sampling across the full range of candidate counts. The conditions behind these gains are consistent as well. Recursion is most effective when applied to early layers, while jointly updating vision and language tokens yields the strongest improvements. The gains even persist under greedy decoding, where repeated calls to the standard model return the same response. Architectural samples also have lower Self-BLEU than temperature samples, indicating lower lexical overlap (Zhu et al., 2018).

Having established what the method does and where it is most effective, we are led to a final and more searching question: *Beyond producing better samples, can architectural sampling actually improve how a model learns?*

To answer this, we use architectural sampling as a drop-in replacement for the temperature-based rollouts in TTRV, a label-free, GRPO-based test-time reinforcement learning method (Shao et al., 2024; Singh et al., 2026). Since TTRV learns directly from the model's own sampled responses, the quality and diversity of these candidates directly shape the resulting update. We find that replacing temperature-based rollouts with architectural rollouts consistently improves accuracy, showing that the benefits of architectural sampling extend beyond candidate generation and pass@k to test-time learning.

To conclude, we summarize the contributions of our work as follows:

- We introduce architectural sampling, a training-free mechanism that reuses layer windows to produce candidates from a frozen VLM without modifying the checkpoint or adding parameters.
- We demonstrate the effectiveness of architectural sampling across five Qwen checkpoints and twelve multimodal benchmarks. At the same candidate budget, it improves pass@9 by an average of 6.58% points over temperature sampling.
- We further show that using architectural candidates as rollouts for label-free test-time reinforcement learning improves accuracy over temperature-sampled rollouts. These results extend the value of architectural sampling beyond candidate coverage, showing that it enables more effective learning from a model's own outputs.

2 Related Work

Architectural sampling sits between test-time scaling and recursive computation. The former spends additional inference-time computation to obtain better outputs; the latter changes the computation performed within the model. We review these two lines in turn.

VLMs and Test-Time Scaling. Test-time scaling seeks more from a fixed checkpoint through repeated generation, aggregation, search, or online adaptation (Snell et al., 2025; Kaya et al., 2026; Singh et al., 2026).

These methods use additional responses in different ways. Self-consistency aggregates independently sampled reasoning paths (Wang et al., 2023). Visual self-ensembling aggregates answers to paraphrases of the same question (Nguyen et al., 2024). VisuoThink and VReST explore reasoning paths with tree search (Wang et al., 2025; Zhang et al., 2025). Test-time augmentation varies the input and aggregates token-level predictions (Kaya et al., 2026), while TTRV turns response frequency and the entropy of the empirical output distribution into rewards for online adaptation (Singh et al., 2026). These methods differ in how they use the responses, but each begins with the alternatives the model has produced.

Temperature and nucleus sampling are common ways to build such candidate pools (Holtzman et al., 2020). They vary token selection, but every response still traverses the same checkpoint and ordered sequence of layers. Candidate coverage is commonly measured with pass@k, the fraction of examples for which at least one of k responses is correct (Chen et al., 2021). Architectural sampling introduces a complementary source of diversity by varying which layer windows are reused and

how often. It remains compatible with stochastic decoding, allowing computational and token-level variation to be combined.

Recursive Architectures and Inference-Time Layer Reuse. Recursive computation has usually served a different goal: deepening or refining a single forward pass. The Universal Transformer applies a shared block repeatedly across depth. Looped Transformers study the resulting computation as a programmable process and as iterative latent reasoning (Dehghani et al., 2019; Giannou et al., 2023; Saunshi et al., 2025). Mixture-of-Recursions learns token-specific recursion depths inside a model trained for recursive computation (Bae et al., 2025).

The same pattern appears in multimodal systems. ReVisE repeatedly updates visual features, answers, and explanations (Ge et al., 2023). ReVisionLLM narrows a temporal search interval over several passes for long-video grounding (Hannan et al., 2025). ReT trains a recurrent cell for multimodal document retrieval (Caffagni et al., 2025). RecursiveVLM builds recursion into continued pretraining and supervised fine-tuning through a learned connector and a monotonic recursion loss (Xu et al., 2026). In these systems, repeated computation is learned for a particular refinement task.

Closer to our setting, recent work rearranges or repeats the layers of pretrained language models after training. CoLa searches, for each input, over skipped, reordered, and repeated layers of a frozen LLM (Li et al., 2025). Inner Loop Inference repeatedly applies a selected block range (Lys et al., 2026). Training-Free Looped Transformers instead uses damped substeps when looping a mid-stack block (Chen et al., 2026). These studies show that selected blocks of a frozen checkpoint can be reused at inference without retraining the base model.

Our work explores a complementary use of recursion as a source of candidate diversity. Each layer reuse configuration defines a distinct computation path for candidate generation, turning repeated computation into a sampling axis. Section 3 formalizes the resulting candidate pool, which we evaluate for solution coverage and use as a source of rollouts for test-time learning.

3 Architectural Sampling

Figure 3 shows how this sampling axis changes the forward pass. Given the same image–prompt pair, every candidate uses the same frozen checkpoint and decoding rule. The standard path applies each decoder layer once. An architectural path instead revisits one contiguous layer window before resuming the original layer order. Varying the window and its recursion depth produces the candidate pool.

3.1 Standard Forward Pass

We consider a pretrained VLM that maps an input $x = (x_v, x_t)$ to a response $y = (y_1, \ldots, y_m)$, where x_v is an image and x_t is a text prompt. The vision encoder and projector map x_v to visual tokens h_v, while the tokenizer and embedding layer map x_t to text embeddings h_t. At decoding step s, the language decoder also receives embeddings $h_{y_{<s}}$ of the tokens generated so far:

$$h_s^{(0)} = [h_v; h_t; h_{y_{<s}}]. \tag{1}$$

For $s = 1$, the generated prefix is empty. The decoder contains L decoder layers $f_1, \ldots, f_L$ with parameters $\theta_1, \ldots, \theta_L$. A standard forward pass applies the layers once in their original order:

$$h_s^{(\ell)} = f_\ell\left(h_s^{(\ell-1)}; \theta_\ell\right), \qquad \ell = 1, \ldots, L. \tag{2}$$

The output head reads the final-position state in $h_s^{(L)}$ to predict y_s, and this process repeats as the response grows. The checkpoint remains fixed throughout candidate generation. Architectural sampling changes only this computation path. To make that change explicit, we omit the step index s and the fixed parameters θ_ℓ below, writing $h^{(0)} = h_s^{(0)}$ for the decoder input at the current step, $f_\ell(\cdot)$ for the frozen layer map, and $h^{(\ell)}$ for its output.

3.2 Reusing a Layer Window

Architectural sampling perturbs the forward pass along two axes: which window of layers is chosen, and how many times it is applied. A block is a contiguous window of layers, and a recursion depth r

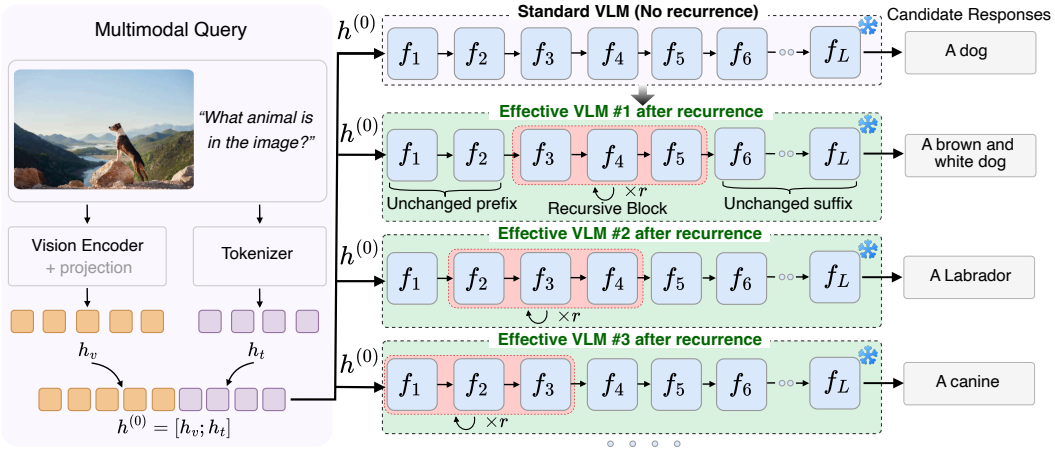


Figure 3: **Architectural Sampling from a Frozen VLM.** (left) The image and prompt form the initial multimodal context, to which generated tokens are appended. (right) The standard path applies every decoder layer once. Each architectural path repeats one contiguous window r times before continuing through the remaining frozen layers in their original order. All paths share frozen weights; changing the window and r yields the candidate pool. Snowflakes mark frozen weights.

re-applies that window in place before the network proceeds:

$$B_{[a:b]} = f_b \circ \cdots \circ f_a, \qquad \big(B_{[a:b]}\big)^{\circ r} = \underbrace{B_{[a:b]} \circ \cdots \circ B_{[a:b]}}_{r \text{ times}}.$$

For a configuration $(B_{[a:b]}, r)$ the forward pass keeps the frozen prefix and frozen suffix untouched and perturbs only the middle stage. Using the same layer maps f_ℓ and hidden states $h^{(\ell)}$ as the standard pass:

$$\begin{aligned} h^{(a-1)} &= f_{a-1} \circ \cdots \circ f_1\big(h^{(0)}\big), && \text{frozen prefix (empty if } a = 1) \\ h^{(b)} &= \big(B_{[a:b]}\big)^{\circ r}\big(h^{(a-1)}\big), && \text{reuse the window } r \text{ times} \\ h^{(\ell)} &= f_\ell\big(h^{(\ell-1)}\big), \quad \ell = b{+}1, \ldots, L, && \text{frozen suffix (empty if } b = L) \end{aligned}$$

Reading the whole pass left to right makes the logic explicit: only the lavender factor and its blue exponent ever change, and the two frozen stages stay fixed:

$$h^{(L)} = \underbrace{f_L \circ \cdots \circ f_{b+1}}_{\text{suffix (fixed)}} \circ \underbrace{\big(B_{[a:b]}\big)^{\circ r}}_{\substack{\text{window} \times r \\ \text{(the two knobs)}}} \circ \underbrace{f_{a-1} \circ \cdots \circ f_1}_{\text{prefix (fixed)}}\big(h^{(0)}\big).$$

Setting $r = 1$ for any window recovers the standard forward pass exactly. Increasing r re-processes the current representation through the same frozen sub-network, while shifting $B_{[a:b]}$ changes *which* computation is repeated. Recursion depth and window location thus induce distinct feature trajectories through identical parameters, producing different outputs from a single checkpoint.

3.3 Building the Candidate Pool

To build a candidate set of size k we take the grid formed by a set of windows $\mathcal{B}$ and a set of recursion depths $\mathcal{R}$, and run one forward pass per pair, i.e. one candidate per cell of the $\mathcal{B} \times \mathcal{R}$ grid:

$$\mathcal{Y} = \big\{\, y_{(B_{[a:b]}, r)} \;:\; B_{[a:b]} \in \mathcal{B},\ r \in \mathcal{R} \,\big\}, \qquad k = |\mathcal{B}| \cdot |\mathcal{R}|.$$

These candidates support inference-time objectives such as pass@k and any setting that requires a pool of self-generated trajectories, including RL for VLMs. Because every configuration uses the same frozen checkpoint and the same decoding temperature, gains reflect *additional candidate diversity induced by architectural variation*. Architectural sampling therefore provides a general candidate-generation primitive, in which the choice of $\mathcal{B}$ and $\mathcal{R}$ sets the size and diversity of the resulting set $\mathcal{Y}$.

4 RESULTS

The construction above turns one frozen checkpoint into a pool of nine responses. We now ask whether this pool contains a correct answer more often than an equal-sized pool drawn by temperature sampling along the standard path.

4.1 EVALUATION SETTINGS

Models and benchmarks. We evaluate five Qwen checkpoints from three model generations: Qwen2.5-VL-7B (Bai et al., 2025b), Qwen3-VL-2B and Qwen3-VL-8B (Bai et al., 2025a), and Qwen3.5-VL-2B and Qwen3.5-VL-4B (Qwen Team, 2026). The main evaluation covers CV-Bench, RealWorldQA, CountBench, MathVista, MMStar, and AI2D (Tong et al., 2024; xAI, 2024; Paiss et al., 2023; Lu et al., 2023; Chen et al., 2024; Kembhavi et al., 2016). We also include GQA, A-OKVQA, ScienceQA, CountQA, MMMU, and BLINK (Hudson & Manning, 2019; Schwenk et al., 2022; Lu et al., 2022; Tamarapalli et al., 2025; Yue et al., 2023; Fu et al., 2024). This set spans visual perception, counting, knowledge, and multimodal reasoning in both multiple-choice and open-ended formats. Further details on the implementation and dataset characteristics are provided in Appendix B.

Candidate generation. For architectural sampling, we use the pool defined in Section 3. Following the one-based indexing used there, the four layer windows are [1:3], [2:4], [3:5], and [4:6]. We generate one response from the standard path and one from each window at recursion depths $r \in \{2, 4\}$, giving nine candidates. Every path is decoded with temperature $T = 0.6$.

The baseline draws nine responses from the standard path, also with $T = 0.6$. Prompts, decoding settings, and answer evaluation are identical, so the comparison matches candidate count and decoding temperature.

Evaluation Protocol. We report all results using the pass@k metric (Shao et al., 2024). Given candidates $\{y_1, \ldots, y_k\}$, where $\mathbf{1}[\cdot]$ is the correctness indicator, we define

$$\text{pass@}k = \mathbb{E}\left[\max_{1 \le i \le k} \mathbf{1}[y_i \text{ is correct}]\right].$$

i.e., a query is solved if at least one candidate is correct. Throughout the main Table 1, we evaluate with $k = 9$ and report pass@9 as a percentage.

4.2 MAIN RESULTS

Table 1 compares architectural sampling with temperature sampling. Across five checkpoints and twelve benchmarks, architectural sampling improves pass@9 by 6.58 percentage points on average. The improvement is consistent across all five checkpoint-level averages and occurs in 55 of the 60 checkpoint–benchmark comparisons. The largest checkpoint-level gains are observed for Qwen3-VL-8B (+12.43 points) and Qwen3.5-VL-4B (+9.13 points), while the smallest gain is +1.10 points for Qwen3.5-VL-2B. The improvement is also broad across tasks, with the largest gains on MMMU (+12.43), MMStar (+11.01), BLINK (+9.28), and RealWorldQA (+8.42). Within each Qwen model family, the larger checkpoint consistently benefits more, suggesting that increased model capacity leaves greater room for alternative computation paths to recover additional correct candidates.

Importantly, this behavior is not specific to the Qwen family. We further evaluate architectural sampling on InternVL3 and Gemma3 across a subset of benchmarks, with results reported in Table 8. Architectural sampling yields average gains of 1.54 and 17.63 percentage points for InternVL3 and Gemma3, respectively. We additionally evaluate the robustness of architectural sampling in Table 7, where we examine the consistency of its gains across repeated runs. Taken together, these results show that the benefits of varying the computation path extend beyond a single model family and remain robust across different architectures, model scales, and multimodal tasks.

Table 1: **Candidate Coverage Across Checkpoints and Benchmarks.** pass@9 for nine temperature samples from the standard path and nine architectural candidates, one from each path, across five Qwen checkpoints and twelve benchmarks. All candidates use $T = 0.6$. The Δ row reports architectural sampling minus standard-path temperature sampling. Avg. is the mean across benchmarks, and the higher score for each checkpoint–benchmark pair is bold.

	CV-Bench	RealWorldQA	CountBench	MathVista	MMStar	AI2D	GQA	A-OKVQA	ScienceQA	CountQA	MMMU	BLINK	Avg.
Qwen2.5-VL-7B													
Temperature	92.46	86.67	97.75	72.60	79.47	90.75	70.15	93.36	92.71	86.36	78.28	87.52	85.67
Architectural	**95.72**	**89.80**	**98.82**	**75.20**	**86.80**	**95.52**	**76.50**	**97.03**	**97.37**	**86.99**	**83.15**	**94.07**	**89.75**
Δ	▲3.26	▲3.13	▲1.07	▲2.60	▲7.33	▲4.77	▲6.35	▲3.67	▲4.66	▲0.63	▲4.87	▲6.55	▲4.07
Qwen3-VL-2B													
Temperature	85.22	72.29	94.62	43.70	53.33	74.85	64.60	85.68	75.66	82.30	35.33	63.43	69.25
Architectural	**89.58**	**85.23**	**96.94**	**50.90**	**60.20**	**76.83**	**70.70**	**88.91**	**81.01**	**85.84**	**48.81**	**69.86**	**75.40**
Δ	▲4.36	▲12.94	▲2.32	▲7.20	▲6.87	▲1.98	▲6.10	▲3.23	▲5.35	▲3.54	▲13.48	▲6.43	▲6.15
Qwen3-VL-8B													
Temperature	88.48	72.94	97.27	58.70	58.93	84.89	64.75	88.65	88.75	76.52	48.56	71.50	75.00
Architectural	**96.40**	**88.10**	**98.83**	**64.10**	**82.73**	**93.48**	**76.20**	**95.72**	**97.97**	**86.05**	**80.27**	**89.28**	**87.43**
Δ	▲7.92	▲15.16	▲1.56	▲5.40	▲23.80	▲8.59	▲11.45	▲7.07	▲9.22	▲9.53	▲31.71	▲17.78	▲12.43
Qwen3.5-VL-2B													
Temperature	91.70	85.75	98.08	**65.60**	73.47	**92.03**	71.25	91.79	**92.02**	82.14	**59.93**	80.20	82.00
Architectural	**94.20**	**92.94**	**98.46**	63.70	**74.27**	85.58	**78.40**	**94.41**	90.68	**87.16**	55.93	**81.46**	**83.10**
Δ	▲2.50	▲7.19	▲0.38	▼1.90	▲0.80	▼6.45	▲7.15	▲2.62	▼1.34	▲5.02	▼4.00	▲1.26	▲1.10
Qwen3.5-VL-4B													
Temperature	68.42	88.50	97.47	64.80	42.60	49.18	72.20	87.34	63.71	**77.34**	30.21	63.05	67.07
Architectural	**91.39**	**92.16**	**98.09**	**66.80**	**58.87**	**65.24**	**76.85**	**88.82**	**75.71**	76.73	**46.32**	**77.43**	**76.20**
Δ	▲22.97	▲3.66	▲0.62	▲2.00	▲16.27	▲16.06	▲4.65	▲1.48	▲12.00	▼0.61	▲16.11	▲14.38	▲9.13

5 Understanding Architectural Sampling

Having established that architectural sampling improves inference-time performance, we now examine it more closely. Unless noted otherwise, all experiments in this section use Qwen3-VL-8B on a five-dataset subset of our benchmarks. With the model and data held fixed, we study the roles of layers and tokens, connect our results to broader questions of temperature and diversity, and conclude with a practical demonstration that further strengthens our claims about its success.

Where should the window be placed? Table 2 moves the repeated window from early to middle and late decoder layers while keeping the candidate budget fixed. Every region improves over temperature sampling, but the gain falls as the window moves deeper. Reusing the early layers gives the highest pass@9 on all five benchmarks, so we use early windows in the main configuration.

Which tokens should be reprocessed? Table 3 compares different token update strategies within the repeated window. Jointly reprocessing visual and language tokens achieves the highest pass@9 on all five benchmarks. Language-only updates retain most of the improvement, while updating only visual tokens or their averaged representation yields smaller gains. We therefore keep the reprocessing of both modalities in our main configuration.

How does architectural sampling interact with temperature? Architectural sampling can be applied at any decoding temperature, including temperature 0, where decoding is fully greedy. As shown in Figure 4a, accuracy remains largely stable across the full range of temperatures we evaluate. This suggests that architectural sampling provides a consistent source of candidate diversity that is largely independent of decoding temperature, maintaining broad coverage of the answer space even at low temperatures.

How diverse are the responses? We measure lexical diversity using $1 - \text{Self-BLEU}$ (Zhu et al., 2018), where higher values indicate lower lexical overlap among candidates. Figure 4b shows a substantially higher mean diversity for architectural sampling (0.51) than for temperature sampling (0.22). Architectural candidates therefore exhibit considerably less lexical overlap and explore a broader range of responses.

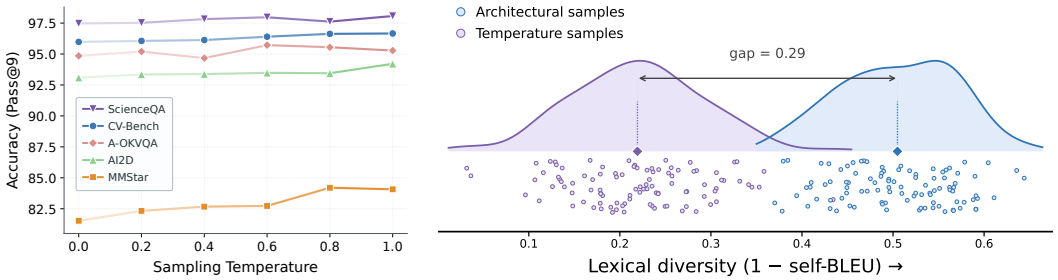
Does architectural sampling outperform standard decoding strategies? Table 4 compares architectural sampling with beam search, top-k, top-p, and temperature sampling at the same candidate

Table 2: **Layer-Window Placement.** pass@9 when the repeated window is placed in early, middle, or late layers of Qwen3-VL-8B. The best setting per benchmark is bold.

	CV-Bench	MMStar	AI2D	A-OKVQA	ScienceQA
Temperature	88.48	58.93	84.89	88.65	88.75
Early layers	**96.40**	**82.73**	**93.48**	**95.72**	**97.97**
Middle layers	93.25	76.13	92.96	92.05	95.04
Late layers	90.83	67.20	88.28	90.57	92.96

Table 3: **Tokens Reprocessed by the Window.** pass@9 when the repeated window updates different token states in Qwen3-VL-8B. The best setting per benchmark is bold.

	CV-Bench	MMStar	AI2D	A-OKVQA	ScienceQA
Temperature	88.48	58.93	84.89	88.65	88.75
Only visual tokens	90.60	65.73	90.85	91.80	94.10
Only language tokens	96.29	81.60	93.42	95.11	96.68
Average visual token	89.61	62.20	88.45	90.04	92.61
All tokens	**96.40**	**82.73**	**93.48**	**95.72**	**97.97**



(a) Effect of decoding temperature on architectural sampling.

(b) Lexical diversity of the two candidate pools.

Figure 4: **Temperature Robustness and Candidate Diversity.** Results on Qwen3-VL-8B over the five benchmarks. (left) Architectural sampling maintains stable pass@9 across decoding temperatures, including greedy decoding ($T = 0$). (right) Architectural candidates have higher $1 - \text{Self-BLEU}$ than temperature samples. Density curves and individual observations show the two distributions, with diamonds marking their means (0.51 vs. 0.22).

Table 4: **Comparison with Standard Decoding strategies.** pass@9 for Qwen3-VL-8B using temperature sampling, beam search, top-k, top-p, and architectural sampling under equal candidate budgets. Architectural sampling is highlighted; the best score per benchmark is bold.

	CV-Bench	MMStar	AI2D	A-OKVQA	ScienceQA
Temperature	88.48	58.93	84.89	88.65	88.75
Beam Search	86.09	52.60	80.78	87.16	86.22
Top-k	89.12	60.27	85.65	89.61	89.74
Top-p	88.91	59.84	85.37	89.28	89.41
Architectural	**96.40**	**82.73**	**93.48**	**95.72**	**97.97**

budget. Architectural sampling achieves the highest pass@9 across all benchmarks in this comparison, indicating stronger candidate coverage than standard decoding strategies. These results show that varying the final output distribution alone provides only limited gains, since all standard decoding strategies sample from outputs produced by the same underlying computation. In contrast, architectural sampling varies the computation itself, leading to substantially greater candidate diversity and, consequently, higher pass@9. This highlights the value of introducing diversity upstream in the computation rather than only at the final decoding stage.

Can architectural candidates improve test-time learning? A diverse candidate pool is valuable not only for inference but also for self-improvement methods that learn from a model's own outputs. To test this, we adopt a label-free reinforcement-learning setup in the style of TTRV (Singh et al., 2026), where the reward is derived from the model's own architectural rollouts rather than ground-truth labels, and we update the model with GRPO (Shao et al., 2024). We evaluate on two backbones, Qwen3-VL-8B and Qwen3.5-VL-2B, across six datasets, comparing standard TTRV (temperature-sampled rollouts) against our variant using architectural rollouts, measured by standard accuracy or pass@1 as shown in Table 5. Architectural sampling consistently improves over vanilla TTRV across backbones and datasets, showing that its induced diversity is not mere noise: it carries a signal correlated with correctness that gradient-based optimization can exploit. This demonstrates that

architectural sampling is useful not only for test-time inference but also as a source of informative self-supervision for test-time training.

Table 5: **Architectural Rollouts for Test-Time Learning.** pass@1 accuracy after TTRV on Qwen3.5-VL-2B and Qwen3-VL-8B. Architectural-rollout cells report the absolute change over temperature-sampled rollouts; the best setting per benchmark is shown in bold.

	CV-Bench	MMStar	AI2D	A-OKVQA	ScienceQA	CountBench
QWEN3.5-VL-2B						
Temperature rollouts	74.60	45.52	75.38	68.04	67.01	93.06
Architectural rollouts	**80.29** ▲ 5.69	**49.17** ▲ 3.66	**76.94** ▲ 1.56	**83.38** ▲ 15.34	**77.33** ▲ 10.32	**95.10** ▲ 2.04
QWEN3-VL-8B						
Temperature rollouts	86.17	53.93	76.61	87.40	84.75	93.17
Architectural rollouts	**86.29** ▲ 0.11	**55.52** ▲ 1.59	**77.80** ▲ 1.17	**87.95** ▲ 0.55	**88.82** ▲ 4.07	**96.93** ▲ 3.76

Changing the computation path remains useful even without decoding noise. It also produces less repetitive candidates, and those candidates improve TTRV when used as rollouts.

6 Limitations and Conclusion

Limitations. Our method comes with two limitations. First, because architectural sampling relies on recursion at its core, it increases the per-candidate forward cost relative to temperature sampling, where every candidate consumes the same resources. We provide an extensive cost analysis in Appendix E and find that, with the careful use of recursion as adopted in our work, the overhead is marginal (roughly 1–2 seconds more, depending on recursion depth, up to depth 4). Nevertheless, this added cost may matter in resource-constrained settings, and we encourage readers to weigh it accordingly. Second, architectural sampling requires access to the internal decoder layers, which precludes its use with closed-weight models served only through an API.

Conclusion. We introduced architectural sampling, a training-free way to generate candidates from different computation paths through a frozen VLM. Each path reuses an early layer window before resuming the original forward pass, turning one checkpoint into a family of candidate generators. At a fixed budget of nine candidates, this improves pass@9 over temperature sampling by 6.58 percentage points on average across our main evaluation. The ablations show that early multimodal reprocessing works best among the tested settings, while coverage remains high under greedy decoding and the resulting candidates have less lexical overlap. When used as rollouts, these candidates also improve test-time learning. Architectural sampling therefore closes the loop on our opening questions: a frozen VLM need not follow a single computation path at test time. Its own computation graph provides a second sampling space, complementary to token-level randomness and available without retraining the checkpoint. This reframes what test-time scaling can mean. We have been scaling test-time compute by asking a model to say more things, when we should have been letting it think in more ways. The weights, it turns out, were never the ceiling. The ceiling was our insistence on using them in only one way.

AI Use Statement

Generative AI tools assisted with language editing, manuscript organization, and LaTeX presentation. They were not used to generate experimental data or determine the reported results. The authors reviewed the resulting text, equations, citations, and presentation, and take responsibility for the final manuscript.

Reproducibility Statement

The main paper and appendix summarize the architectural configurations, candidate budget, decoding settings, model families, benchmarks, and metrics used in our experiments. We will release the implementation and evaluation scripts with the final paper.

REFERENCES

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, et al. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

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*, 2025a.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2.5-VL technical report. *arXiv preprint arXiv:2502.13923*, 2025b.

Davide Caffagni, Sara Sarto, Marcella Cornia, Lorenzo Baraldi, and Rita Cucchiara. Recurrence-enhanced vision-and-language transformers for robust multimodal document retrieval. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9286–9295. IEEE, 2025.

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

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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

Xingyu Fu, Yushi Hu, Bangzheng Li, Yu Feng, Haoyu Wang, Xudong Lin, Dan Roth, Noah A. Smith, Wei-Chiu Ma, and Ranjay Krishna. BLINK: Multimodal large language models can see but not perceive. *arXiv preprint arXiv:2404.12390*, 2024.

Jiaxin Ge, Sanjay Subramanian, Trevor Darrell, and Boyi Li. From wrong to right: A recursive approach towards vision-language explanation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 1173–1185, 2023.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, pp. 11398–11442. PMLR, 2023.

Tanveer Hannan, Md Mohaiminul Islam, Jindong Gu, Thomas Seidl, and Gedas Bertasius. Revisionllm: Recursive vision-language model for temporal grounding in hour-long videos. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19012–19022. IEEE, 2025.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020.

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

Mehmet Onurcan Kaya, Desmond Elliott, and Dim Papadopoulos. Efficient test-time scaling for small vision-language models. In *International Conference on Learning Representations*, volume 2026, pp. 146270–146312, 2026.

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.

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.

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. *arXiv preprint arXiv:2209.09513*, 2022.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255*, 2023.

Jonathan Lys, Vincent Gripon, Bastien Pasdeloup, Axel Marmoret, Lukas Mauch, Fabien Cardinaux, and Ghouthi Boukli Hacene. Inner loop inference for pretrained transformers: Unlocking latent capabilities without training. *arXiv preprint arXiv:2602.14759*, 2026.

Tien-Huy Nguyen, Quang-Khai Tran, and Anh-Tuan Quang-Hoang. Improving generalization in visual reasoning via self-ensemble. In *Computer Vision – ECCV 2024 Workshops*, pp. 289–307. Springer, 2024.

Roni Paiss, Ariel Ephrat, Omer Tov, Shiran Zada, Inbar Mosseri, Michal Irani, and Tali Dekel. Teaching CLIP to count to ten. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

Qwen Team. Qwen3.5: Towards native multimodal agents. Qwen Blog, 2026. URL https://qwen.ai/blog?id=qwen3.5.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, volume 2025, pp. 14855–14881, 2025.

Dustin Schwenk, Apoorv Khandelwal, Christopher Clark, Kenneth Marino, and Roozbeh Mottaghi. A-OKVQA: A benchmark for visual question answering using world knowledge. In *European Conference on Computer Vision*, 2022.

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

Akshit Singh, Shyam Marjit, Wei Lin, Paul Gavrikov, Serena Yeung-Levy, Hilde Kuehne, Rogério Feris, Sivan Doveh, James Glass, and M Jehanzeb Mirza. Ttrv: Test-time reinforcement learning for vision language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 33153–33163, 2026.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 10131–10165, 2025.

Jayant Sravan Tamarapalli, Rynaa Grover, Nilay Pande, and Sahiti Yerramilli. Countqa: How well do MLLMs count in the wild? *arXiv preprint arXiv:2508.06585*, 2025.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Shengbang Tong, Ellis Brown, Penghao Wu, Sanghyun Woo, Manoj Middepogu, Sai Charitha Akula, Jihan Yang, Shusheng Yang, Adithya Iyer, Xichen Pan, et al. Cambrian-1: A fully open, vision-centric exploration of multimodal LLMs. *arXiv preprint arXiv:2406.16860*, 2024.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

Yikun Wang, Siyin Wang, Qinyuan Cheng, Zhaoye Fei, Liang Ding, Qipeng Guo, Dacheng Tao, and Xipeng Qiu. Visuothink: Empowering lvlm reasoning with multimodal tree search. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 21707–21719, 2025.

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

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.

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. *arXiv preprint arXiv:2311.16502*, 2023.

Congzhi Zhang, Jiawei Peng, Zhenglin Wang, Yilong Lai, Haowen Sun, Heng Chang, Fei Ma, and Weijiang Yu. Vrest: Enhancing reasoning in large vision-language models through tree search and self-reward mechanism. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3922–3941, 2025.

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

Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. Texygen: A benchmarking platform for text generation models. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, pp. 1097–1100, 2018.

A Appendix

In this appendix, we present additional experiments and implementation details that provide further insight into our method and complement the results presented in the main manuscript. Section B provides additional implementation details for our experiments. Section C presents additional ablation studies that examine further aspects of our method and provide deeper insights into its effectiveness. Section D presents additional scaling curves corresponding to the results reported in the main tables. Finally, Section E provides the computational cost analysis of our method. All experiments were conducted on a machine equipped with an NVIDIA A100 GPU. To facilitate reproducibility during the review process, we also provide the complete codebase (code.zip) together with detailed execution instructions in README.md. The complete codebase will be publicly released upon acceptance.

B Additional Implementation Details

Inference-time sampling. All checkpoint parameters remain frozen during candidate generation. We use the four inclusive windows [1:3], [2:4], [3:5], and [4:6], together with recursion depths $r \in \{2, 4\}$ and one standard path. This gives nine candidates per input. Each path is decoded with temperature $T = 0.6$. We generate at most 32 new tokens for multiple-choice questions and 256 for open-ended questions. The temperature baseline uses the same limits and draws nine responses from the standard path.

Test-time learning. For the experiment in Table 5, we retain the label-free, GRPO-based TTRV procedure (Shao et al., 2024; Singh et al., 2026) and replace its temperature-sampled rollouts with the architectural candidates. For each training prompt we form the rollout group of 6 from these variants, each decoded at temperature 0.6, and assign the scalar reward by majority vote. No ground-truth labels are therefore used during training. Group-relative advantages are computed with the standard mean/standard-deviation normalization. We train on only 20 samples and evaluate on all remaining held-out data. The policy is adapted with LoRA (rank r=32, α=64, dropout 0.05) applied to the attention and MLP projection layers of the language backbone, with the vision encoder kept frozen. Optimization uses AdamW at a learning rate of 1×10^{-5} in `bf16`, with gradient clipping at 1.0 and a KL penalty (β=0.02) toward a frozen reference policy obtained by disabling the LoRA adapter. We report greedy pass@1.

Dataset Description. To comprehensively evaluate our approach, we curate a diverse set of multimodal benchmarks spanning different aspects of visual understanding and reasoning. The complete list of datasets considered in our evaluation, along with their corresponding characteristics, is provided in Table 6. Our selection is designed to cover both fundamental visual capabilities and higher-level reasoning, allowing us to assess the generalization of our approach across heterogeneous tasks and evaluation settings.

We first consider benchmarks that primarily evaluate *visual perception* and *structured reasoning*. These include AI2D (Kembhavi et al., 2016), ScienceQA (Lu et al., 2022), CV-Bench (Tong et al., 2024), MMStar (Chen et al., 2024), and BLINK (Fu et al., 2024). Together, these benchmarks provide complementary evaluations of models' ability to interpret visual content, understand spatial relationships, reason over structured visual information, and perform vision-dependent tasks.

To examine more specific reasoning capabilities, we include CountBench (Paiss et al., 2023) and CountQA (Tamarapalli et al., 2025), which evaluate *visual counting* under increasingly challenging scene conditions. We further include GQA (Hudson & Manning, 2019), A-OKVQA (Schwenk et al., 2022), and MMMU (Yue et al., 2023) to assess broader *multimodal reasoning*, including compositional understanding and reasoning that may require information beyond the immediate visual content. Finally, MathVista (Lu et al., 2023) and RealWorldQA (xAI, 2024) extend the evaluation toward *mathematical* and *real-world reasoning*. These benchmarks require models to combine visual information with reasoning processes and, in some cases, external knowledge or domain-specific understanding.

Overall, the benchmark suite in Table 6 provides a broad and complementary evaluation of multimodal capabilities. By combining general visual understanding benchmarks with targeted evalua-

Dataset	Dataset Focus	Dataset Type
AI2D (Kembhavi et al., 2016)	Diagram understanding and reasoning	MCQ
ScienceQA (Lu et al., 2022)	Multimodal science QA and reasoning	MCQ
CV-Bench (Tong et al., 2024)	2D/3D visual understanding (spatial relations, counting, depth & distance)	MCQ
MMStar (Chen et al., 2024)	Multimodal perception and reasoning (vision-indispensable)	MCQ
BLINK (Fu et al., 2024)	Core visual perception abilities (relative depth, correspondence, multi-view, etc.)	MCQ (with visual prompts)
CountBench (Paiss et al., 2023)	Object counting	Numeric
CountQA (Tamarapalli et al., 2025)	Object counting in real-world scenes (high density, clutter, occlusion)	Numeric / short-answer
GQA (Hudson & Manning, 2019)	Real-world visual reasoning and compositional QA	Mixed: binary + open-ended
A-OKVQA (Schwenk et al., 2022)	Visual QA with world knowledge	Mixed: MCQ + direct answer
MathVista (Lu et al., 2023)	Mathematical reasoning in visual contexts	Mixed: MCQ + free-form
MMMU (Yue et al., 2023)	Multidisciplinary multimodal understanding and reasoning	Mixed: MCQ + open-ended
RealWorldQA (xAI, 2024)	Real-world spatial understanding	Mixed: MCQ + numeric + open-ended

Table 6: **Statistics of the visual reasoning and VQA datasets used in our experiments.** We report the primary focus and dataset type for each benchmark, covering a diverse range of visual perception, reasoning, counting, mathematical, and real-world understanding tasks.

tions of *counting*, *spatial reasoning*, *mathematical reasoning*, and *higher-level multimodal understanding*, we evaluate our approach across a wide range of challenging visual reasoning scenarios.

C Additional Experiments and Ablations

Robustness. To measure run-to-run variation, Table 7 repeats the Qwen3-VL-8B comparison five times. The standard deviation remains below 0.32 points for both candidate-generation strategies on all five benchmarks.

Table 7: **Run-to-Run Variation.** Mean ± standard deviation of pass@9; the best setting per benchmark is bold.

	CV-Bench	MMStar	AI2D	A-OKVQA	ScienceQA
Temperature	88.48 ±0.11	58.93 ±0.14	84.89 ±0.23	88.65 ±0.11	88.75 ±0.09
Architectural	**96.40** ±0.18	**82.73** ±0.12	**93.48** ±0.31	**95.72** ±0.19	**97.97** ±0.25

Generalization to model families. The main evaluation uses Qwen checkpoints (Bai et al., 2025b;a; Qwen Team, 2026). Table 8 adds InternVL3-2B (Zhu et al., 2025) and Gemma3-4B (Team et al., 2025) on the same five-benchmark subset. Architectural sampling raises the five-benchmark average by 1.54 points for InternVL3-2B and 17.63 points for Gemma3-4B. These results provide an initial test beyond Qwen. Broader model-family coverage remains to be studied.

Table 8: **Generalization Beyond Qwen.** pass@9 on InternVL3-2B and Gemma3-4B. Δ denotes architectural minus temperature sampling; the best score per benchmark is bold.

	CV-Bench	MMStar	AI2D	A-OKVQA	ScienceQA	Avg.
InternVL3-2B						
Temperature	94.62	85.07	**93.55**	94.50	96.28	92.80
Architectural	**95.87**	**87.87**	93.52	**96.16**	**98.31**	**94.35**
Δ	▲ 1.25	▲ 2.80	▼ 0.03	▲ 1.66	▲ 2.03	▲ 1.54
Gemma3-4B						
Temperature	57.96	45.73	72.19	80.00	74.67	66.11
Architectural	**81.27**	**70.13**	**87.26**	**91.88**	**88.15**	**83.74**
Δ	▲ 23.31	▲ 24.40	▲ 15.07	▲ 11.88	▲ 13.48	▲ 17.63

D SCALING CURVES

Figures 2, 5, and 6 trace pass@k as candidates are added to three representative checkpoints, extending Table 1 beyond the fixed budget of nine.

Across most plotted checkpoint–benchmark pairs, temperature sampling begins to plateau earlier, while architectural sampling continues to gain as candidates are added. Because the pattern is not uniform across panels, we treat these curves as descriptive views of the candidate pools.

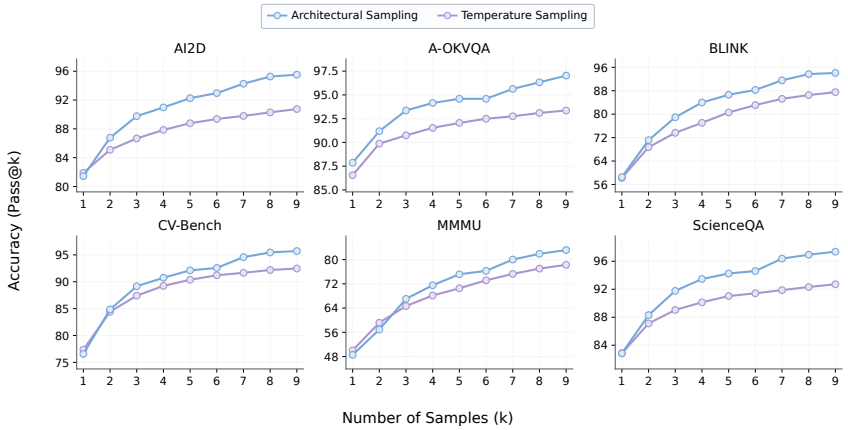


Figure 5: **Candidate Coverage as the Pool Grows: Qwen2.5-VL-7B.** pass@k across six benchmarks as the candidate budget grows for architectural and temperature sampling.

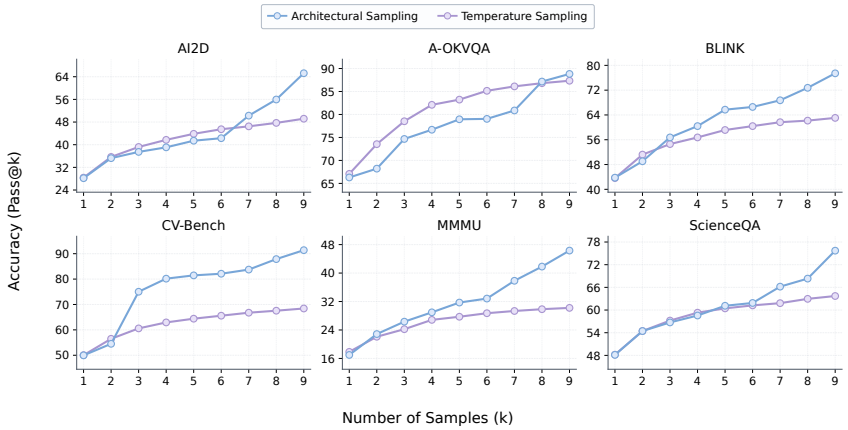


Figure 6: **Candidate Coverage as the Pool Grows: Qwen3.5-VL-4B.** pass@k across six benchmarks as the candidate budget grows for architectural and temperature sampling.

E Computation Cost and Latency

Architectural sampling leaves the parameter count unchanged but traverses some decoder layers more than once. We count these layer applications as an analytic proxy before reporting latency for an 8B-parameter model on one NVIDIA A100.

Cost model. Let a decoder have L layers and a configuration reuse a contiguous window B of size $|B| = b - a + 1$ at depth r. Treating each decoder-layer application equally, the configuration evaluates the prefix once, the window r times, and the suffix once:

$$N_{\text{layers}}(B, r) = \underbrace{(a-1)}_{\text{prefix}} + \underbrace{r\,|B|}_{\text{reused window}} + \underbrace{(L-b)}_{\text{suffix}} = L + (r-1)\,|B|.$$

Normalizing one standard decoder traversal ($r = 1$) to unit cost gives

$$\boxed{C(B, r) = \frac{N_{\text{layers}}(B, r)}{L} = 1 + \frac{(r-1)\,|B|}{L}.}$$

Each additional traversal adds $|B|$ layer applications. The overhead grows with window size and recursion depth and forms a smaller fraction of a deeper decoder. Table 9 instantiates $C(B, r)$ for $L = 32$.

Table 9: **Relative Layer-Application Cost.** For a decoder with $L = 32$, normalized to one standard decoder traversal. Cost grows linearly with window size $|B|$ and recursion depth r; the experimental setting $|B| = 3$ is highlighted.

Window size $\lvert B\rvert$	$r = 1$	$r = 2$	$r = 4$
2	1.00×	1.06×	1.19×
3	1.00×	1.09×	1.28×
4	1.00×	1.13×	1.38×
5	1.00×	1.16×	1.47×

Cost of the full candidate set. Our grid uses $|B| = 3$ for all windows, $|\mathcal{B}| = 4$, and $\mathcal{R} = \{2, 4\}$, along with one standard path at $r = 1$. This gives $k = 9$ candidates. The total decoder-layer cost, normalized to one standard traversal, is

$$\underbrace{1}_{r=1} + \underbrace{4\left(1 + \tfrac{3}{32}\right)}_{r=2} + \underbrace{4\left(1 + \tfrac{9}{32}\right)}_{r=4} = 1 + 4.375 + 5.125 = 10.5.$$

Under this proxy, the nine architectural candidates require 10.5 standard decoder traversals, versus 9.0 for nine temperature samples.

Latency. Table 10 reports the corresponding latency figures. The architectural pool takes 53.8 s, versus 46.1 s for nine temperature samples; its $r = 2$ and $r = 4$ paths add 0.48 s and 1.44 s per candidate.

Table 10: **Wall-Clock Latency.** Per-candidate and total latencies for Qwen3-VL-8B using architectural sampling versus an equal-sized temperature-sampling baseline. The full 9-candidate architectural set is highlighted.

Configuration	# cand.	Cost (×)	Latency (s)	Δ vs. std. (s)
Standard pass (r=1)	1	1.00	5.12	+0.00
Window $\lvert B\rvert$=3, r=2	4	1.09	5.60	+0.48
Window $\lvert B\rvert$=3, r=4	4	1.28	6.56	+1.44
Arch. sampling (total)	9	10.5	53.8	−−
Temp. sampling (total)	9	9.0	46.1	−−