COMPUTE HAS SHAPE: SELF-MORPHING THE INTERNAL COMPUTATION OF FROZEN LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time scaling improves large language models through additional computation, but how to adapt the topology of internal computation to individual inputs remains underexplored. Existing approaches largely fix recurrent or parallel architectures during training, or reconfigure frozen models along a single structural axis. We introduce SELFMORPH, which makes computational shape an input-dependent inference-time decision. It unifies the original forward path (BASE), serial reuse of a layer span (REPEAT), and local branching and merging of perturbed hidden states (BRANCH) in a unified space parameterized by type, location, and intensity. Since they offer complementary benefits rather than uniform dominance, we use a lightweight selector that reads the backbone's depth-wise hidden state trajectory on the prompt to choose a morphology before response generation. We train the selector by distilling utility-based distribution from offline morphing that jointly account for task performance and computational cost. Evaluated on three LLMs spanning 4B–27B, SELFMORPH achieves a 8.06% mean relative improvement on five in-distribution benchmarks and 4.45% on four unseen benchmarks. It achieves the strongest performance in 24 of 27 settings. These findings identify input-adaptive computational topology as a complementary axis of test-time scaling beyond generating more tokens.

1 INTRODUCTION

Scaling test-time computation has become a important way to improve large language models. Chain-of-thought increases serial computation by generating longer reasoning traces (Wei et al., 2022), while self-consistency increases parallel computation by sampling multiple traces and aggregates answers (Wang et al., 2023). These approaches organize additional computation through generated sequences. Computation can also be allocated within the network, raising a further question: *can a frozen LLM adapt its internal computation to each input by choosing among serial reuse, local parallel processing, and its original execution path?*

Recent work provides two foundations for this question. Looped (recurrent-depth) models repeatedly apply shared transformations to deepen serial processing (Dehghani et al., 2019; Geiping et al., 2026; Zhu et al., 2025), while parallel models maintain multiple latent streams and aggregate their outputs (Chen et al., 2026b; You et al., 2026). These studies establish mechanisms for organizing computation within models designed or trained to support them. Another line of work shows that the pretrained layers can be skipped or repeated at inference time (Li et al., 2026; Chen et al., 2026a), demonstrating that frozen weights can support more than one execution path. However, these methods operate within a single axis, and do not place different topologies under a shared cost model. This motivates a broader view: **test-time computation has shape, not only scale**. We investigate whether these computational shapes offer complementary benefits and whether the backbone's own hidden state trajectory can guide their selection.

We formalize computational shape through a parameterized morphology space over a frozen backbone. BASE preserves the original forward path; REPEAT applies a selected layer span repeatedly in series; BRANCH forks perturbed hidden states at the start of the span and merges at the end. The morphology type describes the topology of the computation, while the intervention span and intensity further specify where and how much of the computation is applied. Evaluating these interventions on the same problems reveals complementary effects: REPEAT and BRANCH each solve problems that

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

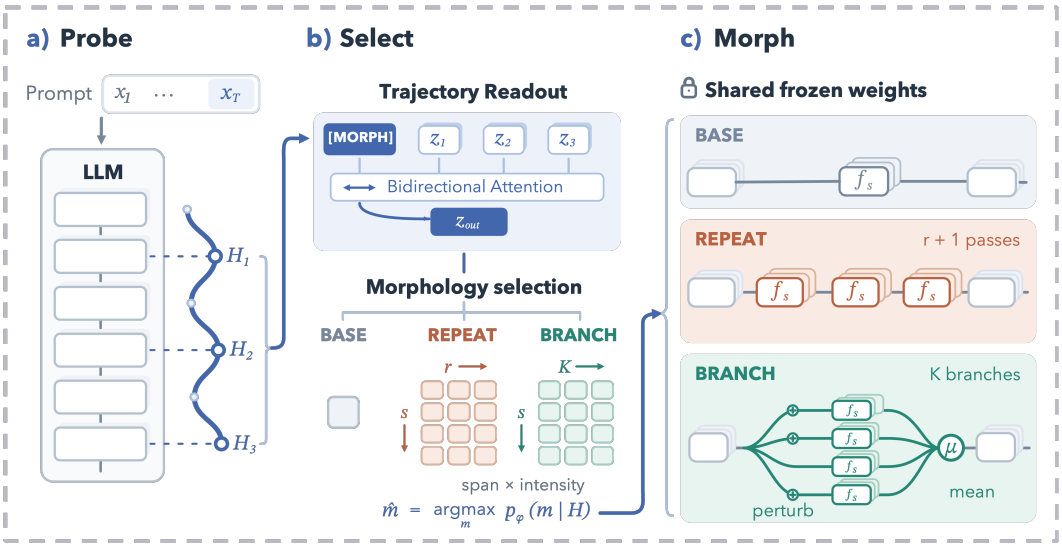


Figure 1: Overview of SELFMORPH. (a) Probe: During prompt prefill along the original forward path, we collect a depth-wise trajectory. (b) Select: A lightweight selector encodes the trajectory and aggregates them through bidirectional attention with a learned token. Before response generation, it selects the highest-probability action. (c) Morph: Three morphology types.

the other does not, and neither uniformly subsumes the other across tasks and backbones. This heterogeneity motivates selecting a morphology for each input rather than applying a single fixed intervention.

As the model encodes the prompt, its hidden state traces a depth-wise trajectory across layers. We introduce SELFMORPH, which reads the trajectory of the LLM itself to decide the shape it ought to adopt. Specifically, SELFMORPH predicts the parameterized morphology, including type, span, and intensity before the first response token is generated. To train the selector, we evaluate candidate actions offline and assign each a utility combining task score and analytic cost. We distill the resulting utility-based teacher distribution into the selector. At inference time, the selected morphology reshapes the model's internal computation throughout generation without evaluating candidate responses. To reduce execution overhead, we reuse prompt computation shared by probing, and share identical cache storage across branch replicas where supported.

Our contributions are as follows.

- We formulate the internal computation of a frozen LLM as a structured choice over BASE, REPEAT, and BRANCH, jointly parameterized by morphology type, span, and intensity. This formulation places serial reuse and local branching within a shared action space over pretrained weights.
- We introduce SELFMORPH, a lightweight attention-based selector that reads the frozen LLM's hidden state trajectory on the prompt and predicts which morphology to apply before generation. We train the selector by distilling utility-based distribution from offline morphing that jointly account for task performance and computational cost.
- We evaluate SELFMORPH on three frozen LLMs across nine benchmarks. Relative to the unmodified backbones, it achieves a mean relative score improvement of 8.06% on five in-distribution benchmarks and 4.45% across on four unseen benchmarks. It also outperforms a per-benchmark selected fixed action in all 15 settings, supporting the value of input-level selection. Extensive ablations also confirm the contributions of morphology diversity, hidden trajectory and attention-based architecture.

2 PARAMETERIZED MORPHOLOGY SPACE

LLMs can be morphed by choosing *what* additional computation to perform (the morphology type), *where* in the layer stack to apply it (the span), and *how much* to add (the intensity). Together they define a parameterized morphology space over a frozen backbone.

2.1 MORPHOLOGY TYPES

Let $\mathcal{M}$ denote a frozen LLM with N layers. A half-open span $s{=}(i, j)$ covers layers $i, i{+}1, \ldots, j{-}1$ with $0 \le i < j \le N$. We denote h_i for the hidden state entering layer i and $f_s = f_{j-1} \circ \cdots \circ f_i$ for the composite transformation over the span. We define three morphology types below, each using the same frozen weights but organizing the computation through s differently.

BASE. The span is traversed once: $h_j^{\text{BASE}} = f_s(h_i)$. No extra computation is added.

REPEAT. The span is traversed $r{+}1$ times in series, where the integer $r \ge 1$ specifies the number of additional passes. Each pass taking the output of the previous one as input:

$$h_j^{\text{REPEAT}} = f_s^{r+1}(h_i), \qquad f_s^{n} := \underbrace{f_s \circ \cdots \circ f_s}_{n \text{ times}}.$$

Successive passes build on earlier updates, providing additional opportunities to refine features before passing the state onward. Each layer retains its built-in normalization throughout the repeated execution. In this sense, REPEAT lets the model "think longer" within a selected layer span, increasing effective depth without introducing new parameters.

BRANCH. The input state is replicated into $K \ge 2$ branches, where K is an integer. At the span entrance, each branches receives a perturbation along an independently sampled Gaussian direction:

$$\tilde{h}_i^{(k)} = h_i + \sigma_k \|h_i\|_2 \, \hat{\epsilon}^{(k)}, \qquad \sigma_k = \frac{k}{K-1}\sigma_{\max}, \quad k = 0, \ldots, K{-}1,$$

where $\hat{\epsilon}^{(k)} = \epsilon^{(k)}/\|\epsilon^{(k)}\|_2$, $\epsilon^{(k)} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mathbf{0}, \boldsymbol{I}_{d_h})$ and d_h is the dimension of hidden state. The hyperparameter $\sigma_{\max}$ controls the maximum relative perturbation magnitude. The schedule preserves one unperturbed input ($\sigma_0 = 0$) and scales the remaining perturbations relative to the input state's norm. Each branches traverses the span independently, and the resulting states are merged by arithmetic averaging:

$$h_j = \frac{1}{K}\sum_{k=0}^{K-1} h_j^{(k)}, \qquad h_j^{(k)} = f_s\big(\tilde{h}_i^{(k)}\big).$$

The merged state is passed to the remaining layers. BRANCH can be seen as exploration in hidden space: controlled perturbations open alternative computational trajectories through the span, creating opportunities to move beyond the original trajectory before the branch outputs are reconciled. The unperturbed branch provides a reference within the ensemble, anchoring this exploration by preserving a direct contribution from the unperturbed input.

Together, these morphologies expose two distinct ways to use additional internal computation: REPEAT deepens the processing of an evolving representation, while BRANCH broadens the computation through perturbation and aggregation.

For each non-BASE morphology, an *action* is a tuple $m = (\tau, s, \theta)$, where $\tau \in \{\text{REPEAT}, \text{BRANCH}\}$ specifies the morphology type, s specifies the location span, and θ is r for REPEAT or K for BRANCH. We represent BASE as a single action with no span or intensity choice.

2.2 CONSTRUCTING THE ACTION SPACE

The morphology space contains $O(N^2)$ candidate spans for each intervention type, each of which can be paired with multiple repetition or branch counts. Moreover, intervention benefits vary substantially across this space, as many span configurations yield negligible gains or degrade performance. We therefore construct a compact action space through screening on two diagnostic datasets, numerical estimation and emotion-intensity prediction, which are separate from the evaluation benchmarks. At a fixed intensity for each type, we evaluate every feasible span and measure its score gain over BASE.

The screening reveals distinct depth preferences across morphology types. Favorable REPEAT spans concentrate at intermediate depths, whereas favorable BRANCH spans tend to occur earlier. This separation is more distinct on the larger backbones. This asymmetry is consistent with the functional

progression across model depth described in prior work: shallow layers build contextual representations, middle layers process reasoning and refine abstract thought, and later layers specialize toward token prediction (Tenney et al., 2019; Lad et al., 2025). Repeating an intermediate span provides additional opportunities to refine features formed by preceding layers. Early branching instead introduces variation before subsequent processing, allowing different computational trajectories to develop within the span before their outputs are merged and passed onward.

To select these candidates, we standardize the mean gains across spans separately for each datasets, type, and backbone. We rank spans by the sum of their standardized gains on the two diagnostic datasets. After evaluating the leading candidates on a larger sample of the same diagnostic tasks, we retain four spans per intervention type. Appendix B provides the complete screening and validation procedure, depth profiles, and selected spans.

For a given backbone, let $\mathcal{S}_\tau$ denote the spans for intervention type τ, and Θ_τ denote its intensity grid. The candidate actions for each type form $\mathcal{A}_\tau = \{\tau\} \times \mathcal{S}_\tau \times \Theta_\tau$. The full action space is

$$\mathcal{A} = \{\textsc{Base}\} \cup \mathcal{A}_{\textsc{Repeat}} \cup \mathcal{A}_{\textsc{Branch}}. \tag{1}$$

We use $\Theta_{\textsc{Repeat}} = \{1, 2, 3\}$ and $\Theta_{\textsc{Branch}} = \{2, 4, 8\}$, yielding 25 actions per backbone. The resulting pool preserves variation in type, location, and intensity while making exhaustive offline evaluation feasible.

3 SELFMORPH

With the morphology space defined, we now have a compact action space $\mathcal{A}$. The remaining question is: which action in $\mathcal{A}$ should be applied for a given state? Enumerating every action incurs an prohibitive computational cost. This section introduces a lightweight selector that makes this decision from the backbone's own hidden states.

3.1 UTILITY-BASED DISTILLATION

Action utility. For an input x and an action $m \in \mathcal{A}$, we define the utility as

$$U(x, m) = \text{score}(x, m) - \lambda\big(C(m) - 1\big), \tag{2}$$

where $\text{score}(x, m) \in [0, 1]$ is the observed task score, $C(m) \geq 1$ is the analytic cost proxy, and $\lambda \geq 0$ controls the penalty for additional computation. The oracle action is

$$m^*(x) = \arg\max_{m \in \mathcal{A}} U(x, m). \tag{3}$$

Evaluating this choice requires generating and scoring candidate responses, so it is available only offline.

Analytic cost. We assign each action a resource penalty based on its intervention width and intensity. For non-BASE actions,

$$C(m) = 1 + \frac{j - i}{N} \cdot k_{\text{eff}}(\tau, \theta),$$

where $j-i$ is the span width, N is the backbone depth, and $k_{\text{eff}}(\tau, \theta)$ is a coefficient determined by the morphology type and intensity (for REPEAT, $k_{\text{eff}} = r$; for BRANCH, k_{eff} accounts for the effective parallel overhead of K branches). We use an analytic proxy rather than wall-clock measurements because runtime depends on hardware, generation length, and batching strategy, all of which introduce noise unrelated to the morphology itself. Appendix E reports the coefficients, their measurement basis, and resource measurements.

Teacher distribution. For each training input, we execute all actions in $\mathcal{A}$ and record their utilities. We construct a teacher distribution that balances utility with entropy:

$$q_T(\cdot \mid x) = \arg\max_{q \in \Delta(\mathcal{A})} \left\{ \sum_{m \in \mathcal{A}} q(m) U(x, m) + T \,\text{Ent}(q) \right\}, \tag{4}$$

where $\Delta(\mathcal{A})$ is the probability simplex, $\mathrm{Ent}(q) = -\sum_m q(m) \log q(m)$, and $T > 0$. The solution is the temperature-scaled softmax (Bogacz, 2017):

$$q_T(m \mid x) = \frac{\exp\big(U(x,m)/T\big)}{\sum_{m' \in \mathcal{A}} \exp\big(U(x,m')/T\big)}. \tag{5}$$

Smaller T concentrates the teacher on higher-utility actions, whereas larger T spreads probability more broadly. This construction gives equal weight to exact ties and represents graded preferences among actions, retaining alternatives that a single oracle label would discard.

Distillation objective. The selector learns to predict the teacher distribution using only $H(x)$. We minimize the cross-entropy

$$\mathcal{L} = -\mathbb{E}_{x \sim \mathcal{D}_{\text{train}}} \left[\sum_{m \in \mathcal{A}} q_T(m \mid x) \log p_\phi(m \mid H(x)) \right]. \tag{6}$$

For the hierarchical prediction heads defined below, this loss decomposes exactly into a type cross-entropy and conditional span–intensity cross-entropies. Each conditional term is weighted by the teacher's probability mass on its morphology type. Thus, the full action distribution supplies supervision to both levels of prediction.

3.2 Trajectory-Conditioned Selection

Hidden trajectory. The selector must decide which morphology to apply before any morphology has been executed. An intuitive approach makes decisions based on the input prompt. But there is a more fundamental signal: the model's own intermediate computations (Azaria & Mitchell, 2023). As LLMs process the input, the hidden state evolves across depth. The depth-wise trajectory exposes how the representation changes through the network, providing features for predicting morphology.

Trajectory encoding. During prompt prefill, we collect the last-token hidden state at L preselected depths and stack these vectors into $H \in \mathbb{R}^{L \times d_h}$. Each slice H_l is pre-normalized using corresponding RMS normalization from the backbone, then mapped through a learnable projection:

$$z_l = W_{\text{enc}} \,\mathrm{PreNorm}(H_l) + e_l, \qquad l = 1, \ldots, L,$$

where $W_{\text{enc}} \in \mathbb{R}^{d_z \times d_h}$ is shared across all L layers and $\{e_l\}$ is a learned embedding table that provides the only positional information in the network.

Depth aggregation. We prepend a learned query token $[\mathrm{MORPH}] \in \mathbb{R}^{d_z}$ to the sequence of layer features. A single bidirectional Transformer block allows this query to attend to all observed depths. Its output at the query position, followed by LayerNorm, gives the shared decision representation $z_{\text{out}} \in \mathbb{R}^{d_z}$.

Hierarchical action prediction. A linear head produces the morphology type logits over $\{\textsc{Base}, \textsc{Repeat}, \textsc{Branch}\}$. For each non-$\textsc{Base}$ type, a two-layer MLP maps z_{out} to $S \times P$ joint distribution over span and intensity, using softmax over $S{\cdot}P$ combinations. The outputs are:

$$p_\phi(m \mid H) = \begin{cases} p_\phi(\textsc{Base} \mid H), & m = \textsc{Base}, \\ p_\phi(\tau \mid H)\, p_\phi(s, \theta \mid \tau, H), & m = (\tau, s, \theta). \end{cases} \tag{7}$$

The factorization shares a trajectory representation across morphology types while modeling span and intensity jointly within each type. Figure 6 summarizes the architecture. Appendix D gives the full specification and training settings.

3.3 Inference: Probe–Select–Morph

Given an input x, SelfMorph selects one action and applies it throughout generation. To reduce the overhead of this process, the implementation reuses computation shared by the probe and the selected execution, and shares eligible cache storage across branch replicas. Inference proceeds in three stages.

Probe. Let $\mathcal{I}$ denote the observed layer indices and $L_{\text{probe}} = 1 + \max \mathcal{I}$. We run a BASE prefill through layers $[0, L_{\text{probe}})$ to construct H. We retain caches and full-prompt hidden states at reusable boundaries, including the probe endpoint and candidate REPEAT span exits reached during the pass.

Select. A selector predicts its highest-probability action from the complete distribution in Equation 7. Selection depends only on H and requires no candidate response generation.

Morph. Execution proceeds according to the selected morphology:

- **BASE.** Resume from the probe endpoint and execute the remaining layers. Probe and morph together perform one ordinary prompt prefill.
- **REPEAT.** For span (i, j), reuse the shared prefix through $\min(L_{\text{probe}}, j)$, completing the first traversal of the span if necessary. Execute r additional traversals, then continue through layers $[j, N)$. Each repeated layer occurrence maintains its own cache history.
- **BRANCH.** Run a separate prefill with K replicas, branching hidden states within the span and averaging them at its exit. We retain the reference batch shape throughout execution to preserve its numerical behavior. Outside the span, eligible standard-attention layers share identical cache rows across replicas; branch-specific histories remain separate.

Prefix reuse avoids recomputing the shared portion of the prompt, while cache sharing reduces storage for replicated histories. The detailed inference optimization and cost measurement are provided in Appendix E.

4 EXPERIMENTS

We evaluate SELFMORPH on three frozen backbones,including Qwen3.5-27B (N=64), Qwen3.5-4B (N=32), and Gemma4-12B (N=48), where N denotes the number of layers. Each backbone uses a 25-action space consisting of BASE, 12 REPEAT and 12 BRANCH actions, where the spans are selected from the screening procedure in Appendix B. Appendix C describes the data splits, prompt protocols, and scoring rules. Appendix D gives the selector architecture and training details. Appendix F presents the analysis of trajectory geometry and per-task dissociation plots.

We compare five strategies: **Base** (LLM itself), **Repeat** (the best REPEAT action in the training set), **Branch** (the best BRANCH action in the training set), **Bench Prior** (the best action in the training set of each specific benchmark), and **SELFMORPH**.

The test set contains five in-distribution benchmarks (BBH (Suzgun et al., 2023), ARC-Challenge (Clark et al., 2018), OpenBookQA (Mihaylov et al., 2018), GSM8K (Cobbe et al., 2021), MATH-500 (Lightman et al., 2024)), and four unseen benchmarks (2WikiMultiHop (Ho et al., 2020), HellaSwag (Zellers et al., 2019), TruthfulQA-MC1 (Lin et al., 2022), GPQA-Diamond (Rein et al., 2023)). All reported numbers are computed on these sets, which are held out from both training and validation. Each benchmark uses its own scoring rule (exact-match accuracy for multiple-choice and arithmetic tasks, token-level F1 for extractive QA). The mean score metric is computed by averaging per-sample scores across test samples. For the main results in Table 1, all strategies are allowed to generate their complete responses without length limit, and scoring is performed on the completed responses. We separately examine a 32-token generation limit in Table 2 to evaluate early-answer performance: whether a correct answer is produced within this window. Under this protocol, a response that has not yet stated its answer receives no credit, even if its continuation would provide the correct answer.

4.1 MAIN RESULTS

Table 1 compares SELFMORPH with the unmodified backbone and fixed morphology strategies. SELFMORPH improves over Base in most evaluated settings and outperforms Bench Prior in every backbone–benchmark pair. Because Bench Prior selects one action per benchmark using the training set, this comparison supports the value of input-dependent selection beyond a benchmark-level fixed strategy. The gains are uneven across backbones and tasks, with the largest improvements on BBH and ARC-Challenge for Qwen3.5-27B and smaller changes on Gemma4-12B.

Table 1: Scores on five benchmarks, evaluated on completed responses. Bold marks the highest score per row. Gain is relative improvement of SELFMORPH over Base.

Backbone	Benchmark	Base	Repeat	Branch	Bench Prior	**SELFMORPH**	Gain
Qwen3.5-27B	BBH	0.5105	0.6632	0.4474	0.6632	**0.6895**	+35.06%
	GSM8K	0.4120	0.4280	0.2720	0.4120	**0.4360**	+5.83%
	MATH-500	0.5400	0.4800	0.4600	0.5300	**0.5600**	+3.70%
	ARC-Challenge	0.6200	0.7667	0.6733	0.7733	**0.7800**	+25.81%
	OpenBookQA	0.8267	0.8333	0.7400	0.8333	**0.8400**	+1.61%
Qwen3.5-4B	BBH	0.4105	0.4421	0.3789	0.4526	**0.5211**	+26.94%
	GSM8K	0.1720	0.1600	**0.1800**	0.1720	**0.1800**	+4.65%
	MATH-500	0.3300	0.2800	0.3000	0.3100	**0.3400**	+3.03%
	ARC-Challenge	0.7867	0.7733	0.7467	0.7467	**0.8000**	+1.69%
	OpenBookQA	0.7867	**0.7933**	0.7400	0.7333	**0.7933**	+0.84%
Gemma4-12B	BBH	0.5947	0.6000	0.5789	0.5947	**0.6105**	+2.66%
	GSM8K	0.2320	0.1920	0.2240	0.2320	**0.2360**	+1.72%
	MATH-500	0.2300	0.2000	0.2300	0.2300	**0.2400**	+4.35%
	ARC-Challenge	0.7733	0.7667	**0.7800**	0.7667	0.7733	+0.00%
	OpenBookQA	0.8800	0.8867	0.8867	0.8800	**0.9067**	+3.03%

Table 2: Short-budget scores on Qwen3.5-27B across three benchmarks. Generation is limited to 32 tokens, so a correct answer must be produced and extracted within this window. This supplementary evaluation characterizes performance under an early-answer constraint.

Benchmark	Base	Repeat	Branch	Bench Prior	**SELFMORPH**	Gain
BBH	0.2000	0.5105	0.1684	0.5105	**0.6053**	+202.65%
ARC-Challenge	0.3667	0.7333	0.5000	0.7333	**0.7467**	+103.63%
OpenBookQA	0.5400	0.7867	0.6267	0.7867	**0.7933**	+46.91%

Response behavior under a short output budget. We also observe an interesting phenomenon in Qwen3.5-27B. Despite the direct-answer prompts, the unmodified model often produces explanatory text before stating its choice. Under morphology intervention, particularly REPEAT, responses tend to state the answer more directly. This suggests that changing internal execution can affect the timing of answer production as well as task performance. Table 2 shows that SELFMORPH achieves higher scores when a correct answer must be extracted in a short window, providing an additional view.

4.2 GENERALIZATION TO UNSEEN BENCHMARKS

Table 3 evaluates each backbone's selector on four benchmarks excluded from selector training, without retraining or refitting. SelfMorph improves over Base in most settings, indicating that the learned selection policy can transfer beyond its training benchmarks.

4.3 ABLATION STUDIES

We examine two central design choices: the composition of the action space and the representation used for selection. By default, all ablations are conducted on three backbones and short-budget protocol. All remaining hyperparameters kept fixed unless otherwise noted. The mean score metric denotes the per-sample average score. Additional ablations of selector architecture, capacity, target temperature, and cost coefficient are provided in Appendix G.

Action space composition. We train selectors on three nested pools: 3 actions (BASE and one action per intervention family), 9 actions (BASE and four spans per family at one intensity), and the full 25 actions (three intensities per span). For the 3-action pool, the span for each intervention family is selected by mean training-set utility. We also evaluate two 13-action pools, each containing BASE and all 12 actions from either REPEAT or BRANCH. Table 4 reports selector performance and the per-sample oracle ceiling for each pool. Expanding the mixed pool improves performance on both Qwen backbones, whereas Gemma matches its 9-action score with 25 actions. The 9-action

Table 3: Scores on four unseen benchmarks. Bold marks the highest score per row.

Backbone	Benchmark	Base	Repeat	Branch	SELFMORPH	Gain
Qwen3.5-27B	2WikiMultiHop	0.7463	0.7483	0.6624	**0.7708**	+3.28%
	HellaSwag	0.9100	**0.9500**	0.8700	0.9300	+2.20%
	TruthfulQA-MC1	0.7900	**0.8000**	0.7300	**0.8000**	+1.27%
	GPQA-Diamond	0.5800	0.5500	0.4000	**0.6000**	+3.45%
Qwen3.5-4B	2WikiMultiHop	0.5643	0.5527	0.5757	**0.5833**	+3.37%
	HellaSwag	0.8500	0.8800	0.8500	**0.8900**	+4.71%
	TruthfulQA-MC1	**0.6700**	0.6500	0.6600	**0.6700**	+0.00%
	GPQA-Diamond	0.4400	0.4600	0.3800	**0.4700**	+6.82%
Gemma4-12B	2WikiMultiHop	0.1381	0.1606	0.1500	**0.1639**	+18.68%
	HellaSwag	0.4300	0.3900	0.4500	**0.4600**	+6.98%
	TruthfulQA-MC1	**0.5200**	**0.5200**	0.5100	0.4800	-7.69%
	GPQA-Diamond	0.2900	0.2600	0.3100	**0.3200**	+10.34%

Table 4: Action space size ablation. Nested pools ($3 \subset 9 \subset 25$) and two 13-action pools are compared. "Oracle" is the per-sample upper bound achievable within that pool.

Backbone	Pool	Mean Score	Oracle
Qwen3.5-27B	3 actions	0.5786	0.6381
	9 actions	0.5869	0.7000
	13 actions (re)	0.5964	0.7202
	13 actions (br)	0.4524	0.6321
	25 actions	0.6083	0.7619
Qwen3.5-4B	3 actions	0.4738	0.5286
	9 actions	0.4833	0.5571
	13 actions (re)	0.4810	0.5524
	13 actions (br)	0.4786	0.5607
	25 actions	0.4964	0.6107
Gemma4-12B	3 actions	0.5310	0.5583
	9 actions	0.5369	0.6440
	13 actions (re)	0.5321	0.6750
	13 actions (br)	0.5274	0.6583
	25 actions	0.5369	0.7071

Table 5: Selector input ablation. "Hidden trajectory (n)" denotes a hidden trajectory composed of n layer.

Input	Mean Score
Prompt embedding	0.5714
Single hidden state	0.5940
Hidden trajectory (3)	0.5964
Hidden trajectory (5)	0.5952
Hidden trajectory (10)	0.6083
Prompt embedding	0.4643
Single hidden state	0.4845
Hidden trajectory (3)	0.4905
Hidden trajectory (5)	0.4952
Hidden trajectory (10)	0.4964
Prompt embedding	0.5321
Single hidden state	0.5286
Hidden trajectory (3)	0.5333
Hidden trajectory (5)	0.5345
Hidden trajectory (10)	0.5369

mixed pool outperforms the larger single-family pools in most comparisons, supporting the value of access to both morphology families over additional intensity choices within one family. However, the oracle–selector gap grows as the mixed pool expands: additional actions increase the available headroom faster than the selector captures it.

Selector input. We compare prompt embeddings, a single intermediate-layer hidden state, uniformly subsampled 3- and 5-layer trajectories, and the default 10-layer trajectory (Table 5). Almost every hidden state condition outperforms the prompt embedding, suggesting that the model's intermediate computation provides stronger signal about morphology than the surface text. The full trajectory achieves the highest score among the tested representations on all three backbones. This supports using multiple depths of the backbone's prompt computation to inform morphology selection.

4.4 ANALYSIS

Trajectory features and morphology preference. To visualize the relationship between hidden trajectory patterns and preferred morphology, we examine whether trajectories differ between samples whose oracle actions belong to distinct families. Figure 2 partitions samples into four categories: *base-hopeless* (all actions defeat), *base-enough* (BASE has the highest utility), *repeat-best*, and *branch-best*. Two features are computed from the trajectory window, before any morphology is applied: (a) the relative hidden state update between the last two observed layers, and (b) the minimum cosine similarity of adjacent observed layers across the window.

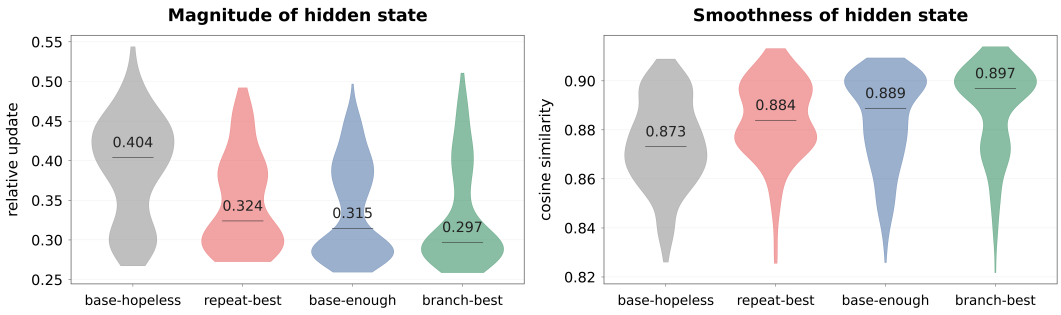


Figure 2: The trajectory features before morphing. Four groups are defined by which family the per-sample oracle selects: base-hopeless (oracle is BASE but sample is unsolved), repeat-best, base-enough (BASE and already correct), branch-best. (a) Relative hidden-state update near the readout point. (b) Minimum cosine similarity across the trajectory window.

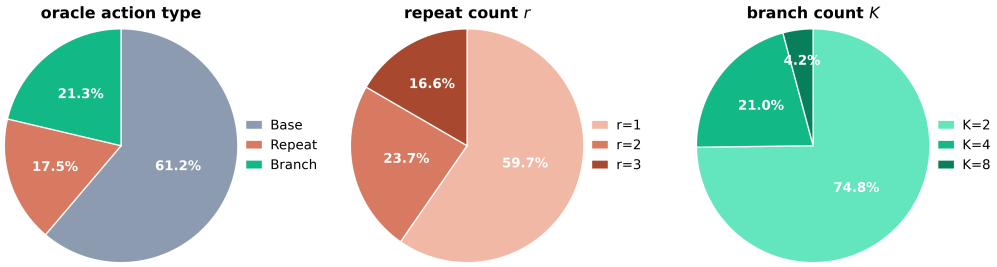


Figure 3: Oracle action distribution. Left: morphology type breakdown. Center: intensity distribution within REPEAT. Right: intensity distribution within BRANCH.

The group summaries exhibit a consistent ordering. Relative updates decrease from base-hopeless through repeat-best and base-enough to branch-best, while cosine similarity increases in the same order. Thus, the oracle groups differ in how strongly their representations change across the observed depth window. We interpret this pattern through a hypothesis about the role of additional computation. Serial reuse may help when further transformations of an evolving representation are useful; perturbation and aggregation may help when a relatively settled representation benefits from exploring nearby states. This interpretation is consistent with the greater trajectory activity of repeat-best relative to branch-best. It motivates a distinction between computational stability and computational sufficiency: the branch-best group has relatively stable trajectories despite benefiting from intervention, while the most active group remains unsolved under the evaluated protocol.

Oracle action distribution. Figure 3 shows the oracle-optimal action distribution on Qwen3.5-27B. The majority of samples are best served by BASE. This group contains both inputs that the backbone already answers correctly and inputs that no morphology can rescue, so BASE wins by incurring zero extra cost. For both REPEAT and BRANCH, the settings with the lowest intensity receive the largest share of selections. And a non-trivial set of problems demands larger intervention intensities, as higher intensity tackle the problems where low intensity fails.

5 CONCLUSION

We introduced SELFMORPH, a framework that makes the internal computation of frozen LLMs an input-dependent choice over the original forward path, serial layer reuse, and local branching. Trained through utility-based distillation of offline interventions, a lightweight selector reads the backbone's prompt trajectory to choose the morphology type, span, and intensity before generation. Across three backbones and nine benchmarks, SELFMORPH demonstrates the value of input-dependent selection and transfers to unseen benchmarks without retraining. Ablations support the contributions of morphology diversity, depth-wise trajectory information, and the architecture components. These findings establish computational shape as a useful dimension of inference-time adaptation.

AI USE STATEMENT

AI tools were used to assist with polishing the manuscript and coding assistance. The authors take full responsibility for the manuscript, code, and the accuracy and integrity of the reported results.

REFERENCES

Amos Azaria and Tom Mitchell. The internal state of an llm knows when it's lying. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 967–976. Association for Computational Linguistics, 2023.

Rafal Bogacz. A tutorial on the free-energy framework for modelling perception and learning. *Journal of Mathematical Psychology*, 76:198–211, 2017.

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

Mouxiang Chen, Binyuan Hui, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Jianling Sun, Junyang Lin, and Zhongxin Liu. Parallel scaling law for language models. *Advances in Neural Information Processing Systems*, 38:118958–118998, 2026b.

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.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

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

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. *Advances in Neural Information Processing Systems*, 38: 41340–41391, 2026.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020.

Vedang Lad, Jin Hwa Lee, Wes Gurnee, and Max Tegmark. Remarkable robustness of llms: Stages of inference? *Advances in Neural Information Processing Systems*, 38:130050–130083, 2025.

Ziyue Li, Yang Li, and Tianyi Zhou. Skip a Layer or Loop It? Learning Program-of-Layers in LLMs. In *Forty-third International Conference on Machine Learning (ICML)*, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. In *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 3214–3252, 2022.

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.

David Noel Ng. Llm neuroanatomy: How i topped the llm leaderboard without changing a single weight. Blog post, March 2026.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, 2023.

Yao Tang, Li Dong, Yaru Hao, Qingxiu Dong, Furu Wei, and Jiatao Gu. Multiplex thinking: Reasoning via token-wise branch-and-merge. *arXiv preprint arXiv:2601.08808*, 2026.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4593–4601, 2019.

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. *International Conference on Learning Representations*, 2023.

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

Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. Softcot++: Test-time scaling with soft chain-of-thought reasoning. *arXiv preprint arXiv:2505.11484*, 2025.

Runyang You, Yongqi Li, Meng Liu, Wenjie Wang, Liqiang Nie, and Wenjie Li. Parallel test-time scaling for latent reasoning models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 44703–44717, 2026.

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.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

A Related Work

Recurrent computation. Recurrent-depth (looped) Transformers organize computation through repeated application of shared transformations. Universal Transformers introduce recurrence across depth (Dehghani et al., 2019); subsequent work trains models with variable unrolled lengths for inference-time scaling (Geiping et al., 2026) and extends this paradigm to large-scale pretraining (Zhu et al., 2025). These models are trained to support recurrent execution. Another line reconfigures pretrained layers while keeping the backbone frozen. PoLar learns input-dependent programs that skip or repeat layer modules (Li et al., 2026), while training-free looped Transformers repeat middle-layer blocks and stabilize their updates without parameter learning (Chen et al., 2026a). SelfMorph shares the goal of adapting execution over pretrained weights. Its focus is the joint selection of serial reuse and local branching, including their spans and intensities, using the backbone's prompt trajectory to predict which intervention is useful.

Parallel computation. Parallel methods expand computation through multiple latent streams. Chen et al. (2026b) learn input transformations and an aggregation mechanism for parallel model executions. SoftCoT++ constructs multiple latent candidates at the token or embedding interface (Xu et al., 2025), while Multiplex Thinking branches over sampled tokens and merges their embeddings during continuous reasoning (Tang et al., 2026). Other approaches sample continuous reasoning trajectories from a latent-reasoning backbone and train a reward model to select among them (You et al., 2026). These methods differ in where streams diverge, how they are combined, and which components are learned. SelfMorph implements branching within a selected layer span of a frozen backbone, using Gaussian perturbations and arithmetic averaging without learning the perturbation or merge operation. Its central question is when this local branching is preferable to serial layer reuse or the original forward path.

B Span Screening

Before fixing the action space $\mathcal{A}$ (Section 2.1), we screen candidate spans at a fixed intensity for each intervention type to identify favorable depth regions for Repeat and Branch. This screening is a diagnostic pass run on a small, fast probe set that is separate from the formal evaluation benchmarks.

B.1 Setup

Backbones. We screen three frozen backbones: Qwen3.5-27B (N=64, d_h=5120), Qwen3.5-4B (N=32, d_h=2560), and Gemma4-12B (N=48, d_h=3840), where N is the backbone depth and d_h is the dimension of hidden state. Every weight parameter is in bfloat16 precision.

Probe set. Screening over thousands of span candidates requires a probe set that is both cheap to evaluate and informative about general model capability. Following by Ng (2026), we use two diagnostic datasets, each containing 16 samples. The first presents large-integer estimation problems (square roots, cube roots, multi-digit products) scored by relative numerical error with a digit-length correction. The second presents emotion-intensity prediction items in an EQ-Bench format, scored by absolute deviation from reference. The two tasks are cognitively orthogonal: one tests abstract numerical intuition, the other tests social and affective reasoning. Using both provides a broader basis for screening than either diagnostic task alone. Both datasets produce short outputs (a single number or a few scores), which keeps each generation fast enough for exhaustive enumeration.

For the numerical estimation probe, the scorer extracts all integers from the generated text, computes for each a relative error against the reference answer, and applies a digit-length correction that penalizes guesses whose number of digits differs from the reference. The score for a single extraction is $(1-\text{relative error})\times(1-\text{padding fraction})$, clamped to $[0,1]$, and the sample score is the maximum over all extracted integers. For the emotion-intensity probe, the scorer parses four predicted intensity values (on a 0–10 scale) from the generated text, computes their summed absolute deviation from the reference, and normalizes by the maximum possible deviation (40). Inputs are formatted as raw-text continuations without a chat template. Generation uses greedy decoding.

Scan grid. For each backbone we enumerate every half-open span (i, j) with $0 \leq i < j \leq N$. REPEAT is tested at $r{=}1$ and BRANCH at $K{=}4$ with $\sigma_{\max}{=}1.0$, using the same perturbation and hidden-space mean merge described in Section 2.1. The screening implementation excludes $j{=}N$ for BRANCH because its merge hook requires a layer at position j. The full grid contains $N(N{+}1)/2$ spans for REPEAT and $N(N{+}1)/2 - N$ for BRANCH, yielding 4097 / 1025 / 2305 actions per sample for the 64 / 32 / 48-layer backbones. All grids were executed to completion with zero missing entries. Scan parameters are identical across all three backbones.

B.2 HEATMAPS

For a fixed morphology type and screening intensity, each cell (i, j) shows the mean score gain over BASE on the 16 samples in the diagnostic dataset $\mathcal{D}_{\text{probe}}$:

$$\Delta(i, j) = \frac{1}{16} \sum_{x \in \mathcal{D}_{\text{probe}}} \left[\text{score}_{(i,j)}(x) - \text{score}_{\text{BASE}}(x) \right].$$

Figures 4 and 5 show the results, with rows corresponding to REPEAT and BRANCH and columns to the three backbones. Red indicates improvement and blue indicates degradation.

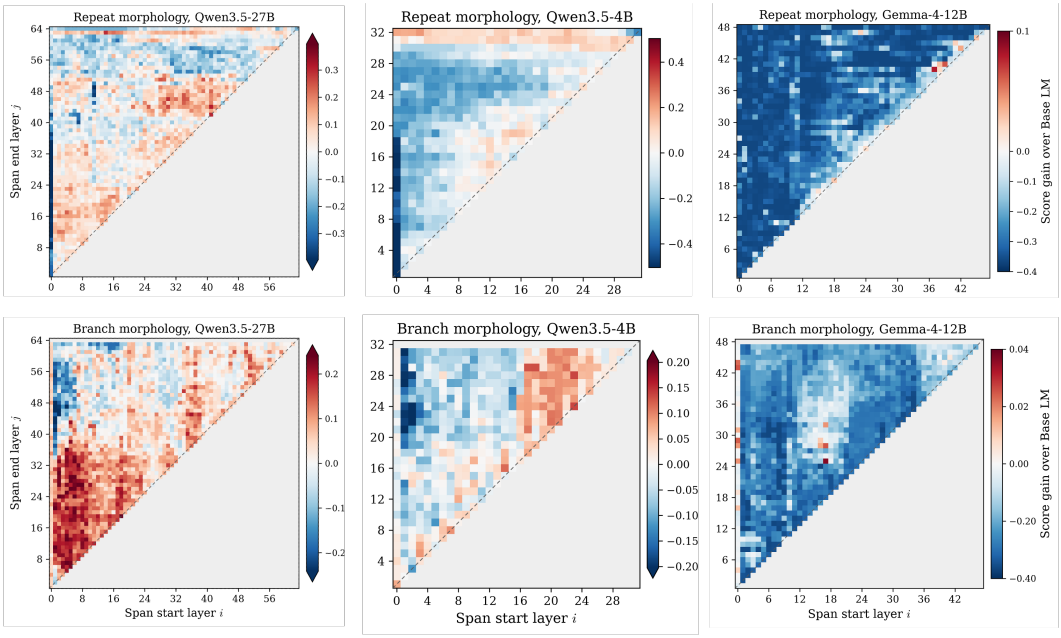


Figure 4: Score gain $\Delta(i, j)$ on the numerical estimation probe. Top: REPEAT ($r{=}1$). Bottom: BRANCH ($K{=}4$). Columns left to right: Qwen3.5-27B ($N{=}64$), Qwen3.5-4B ($N{=}32$), Gemma4-12B ($N{=}48$). The x-axis is span start i, the y-axis is span end j, and only the upper triangle ($i < j$) is valid.

B.3 OBSERVATIONS

Intervention effects depend on span location. Many spans produce negative gains, including spans that begin in the earliest layers. The affected regions vary across morphology types, backbones, and diagnostic tasks, motivating explicit screening.

REPEAT gains concentrate at mid-depth. On all three backbones, the positive region for REPEAT occupies a band in the mid-to-late depth range. This is the depth range where layers iteratively refine task-relevant features (Tenney et al., 2019; Lad et al., 2025), and composing f_s with itself continues to advance the representation (Section 2.1). The pattern holds across both probe datasets and all three model scales.

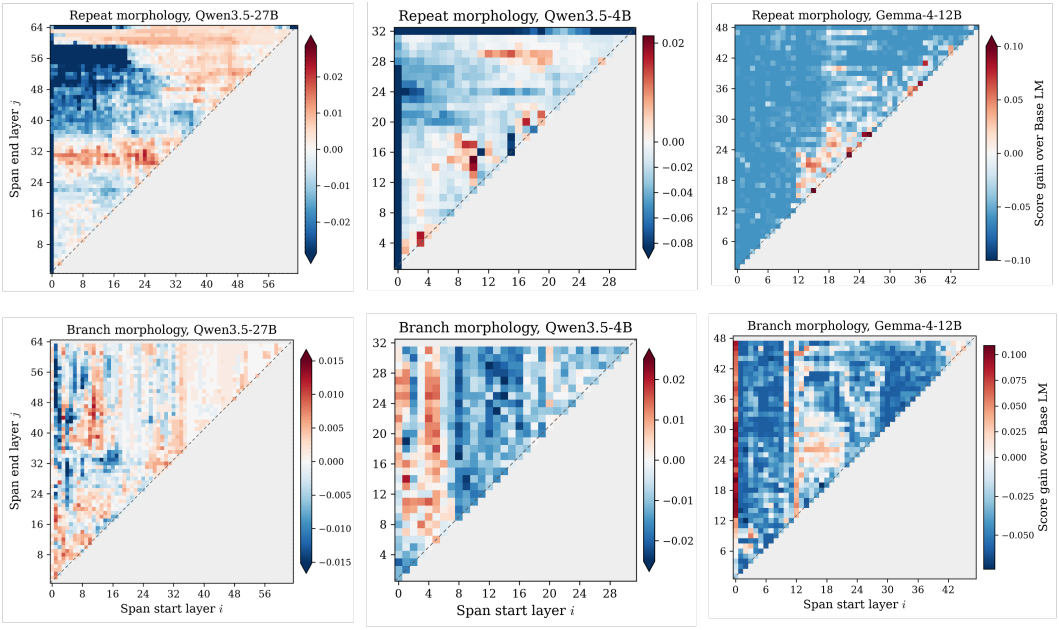


Figure 5: Score gain $\Delta(i, j)$ over BASE on the emotion-intensity probe. Top: REPEAT (r=1). Bottom: BRANCH (K=4). Columns left to right: Qwen3.5-27B (N=64), Qwen3.5-4B (N=32), Gemma4-12B (N=48). The x-axis is span start i, the y-axis is span end j, and only the upper triangle ($i < j$) is valid.

BRANCH gains concentrate at shallow depth. On Qwen3.5-27B, the positive region for BRANCH occupies the lower-left corner of the heatmap (small i, moderate j), with 72% of all BRANCH spans producing a positive delta on the numerical probe. This is consistent with the shallow layers still holding a malleable representation (Tenney et al., 2019; Lad et al., 2025): a perturbation introduced there can still be resolved into a genuinely different downstream computation. On the smaller backbones (4B and 12B), the positive region for BRANCH is broader and extends into mid-depth spans, with the numerical-probe peak at $(23, 28)$ for 4B and $(17, 25)$ for Gemma. This broader spread is consistent with the observation that smaller models have less cleanly separated functional stages: encoding, refinement, and decoding overlap more across depth, so the depth at which branching pays off is less sharply localized. The overall pattern remains that BRANCH gains start shallow and taper off toward the output end of the stack, while REPEAT gains concentrate in the middle.

The two probe datasets rank spans differently. The Spearman rank correlation between the numerical and emotion probes is close to zero on the Qwen backbones ($\rho \approx -0.06$ and $+0.06$) and moderate on Gemma ($\rho = +0.63$). A span that helps on one probe does not reliably help on the other. We therefore rank candidates by a balanced criterion: for each probe, the raw mean Δ of every span is standardized to zero mean and unit variance (z-score), and the two z-scores are summed.

B.4 FROM SCREENING TO ACTION SPACE

This screening step produces a set of candidate options and the selector subsequently chooses an action for each input. Candidate spans are ranked by the balanced z-score (restricted to widths $j-i \leq 10$ to keep analytic cost moderate) and then validated on a larger sample of the same two probe tasks, scored with the procedures described above. Each backbone selects its own spans from its own scan.

Table 6 lists the spans selected for each backbone. The selected REPEAT spans lie near the middle of Qwen3.5-27B, toward the end of Qwen3.5-4B, and across intermediate and later depths of Gemma4-12B. The selected BRANCH spans occur early in Qwen3.5-27B and cover a wider depth range in the smaller backbones.

Table 6: Layer spans in the final action space $\mathcal{A}$ for each backbone. Each span is paired with three intensity levels ($r \in \{1, 2, 3\}$ for REPEAT, $K \in \{2, 4, 8\}$ for BRANCH).

Type		Qwen3.5-27B	Qwen3.5-4B	Gemma4-12B
REPEAT	Span 1	$(33, 35)$	$(30, 31)$	$(36, 37)$
	Span 2	$(31, 35)$	$(28, 31)$	$(26, 27)$
	Span 3	$(30, 36)$	$(24, 31)$	$(15, 16)$
	Span 4	$(26, 35)$	$(24, 30)$	$(25, 27)$
BRANCH	Span 1	$(1, 10)$	$(23, 28)$	$(0, 9)$
	Span 2	$(4, 7)$	$(3, 8)$	$(0, 10)$
	Span 3	$(3, 8)$	$(3, 7)$	$(3, 7)$
	Span 4	$(5, 7)$	$(22, 26)$	$(16, 26)$

C ORACLE COLLECTION AND DATA SPLITS

The selector in SELFMORPH is trained on per-sample utility values obtained by running every action in $\mathcal{A}$ on every training input and recording the result. This appendix describes the datasets, prompt protocols, scoring rules, and collection process behind those measurements.

C.1 BENCHMARK DESCRIPTIONS

We evaluate on nine benchmarks. Five benchmarks are used for training and in-distribution evaluation, and four additional benchmarks are held out entirely from training for zero-shot generalization assessment.

BBH (Suzgun et al., 2023) is a curated subset of BIG-Bench tasks where language models fall below average human performance.

ARC-Challenge (Clark et al., 2018) contains grade-school science questions filtered to those that simple retrieval and co-occurrence methods answer incorrectly.

OpenBookQA (Mihaylov et al., 2018) poses science questions that require combining a provided fact with broader commonsense knowledge.

GSM8K (Cobbe et al., 2021) consists of grade-school mathematical word problems requiring multi-step arithmetic.

MATH-500 (Lightman et al., 2024) samples 500 competition-level mathematics problems covering algebra, geometry, number theory, and combinatorics.

2WikiMultiHop (Ho et al., 2020) is a multi-hop extractive QA task where each question requires combining information from at least two Wikipedia paragraphs.

HellaSwag (Zellers et al., 2019) tests commonsense reasoning by asking the model to choose the most plausible continuation of a sentence from four candidates.

TruthfulQA-MC1 (Lin et al., 2022) evaluates whether models can identify the single truthful answer among plausible distractors. Each question has a variable number of candidates (2 to 13).

GPQA-Diamond (Rein et al., 2023) contains graduate-level science questions (physics, chemistry, biology) written by domain experts under an adversarial "Google-proof" protocol that discards questions answerable by web search.

C.2 DATASETS AND SPLITS

We evaluate on five in-distribution benchmark units drawn from three domains: ARC-Challenge and OpenBookQA (science multiple-choice), GSM8K and MATH-500 (mathematics), and BBH (reasoning). The four unseen benchmarks use separate frozen sample lists and are evaluated with the production selector without any refitting.

Table 7 reports the in-distribution sample counts. The three splits serve distinct roles: the training set supplies the oracle utilities for selector training, the validation set (the union of two equal-sized sub-pools) is used exclusively for early stopping within a training run, and the test set is held out from both training and early stopping. The test set is never seen during any part of the training process, and the numbers reported in Section 4 are computed on it. Samples are drawn by stratified sampling over subtasks or dataset splits with a fixed random seed `20260810`. All three backbones use the same sample manifest.

Table 7: Sample counts per benchmark and split. All three backbones use the same samples, drawn from a single frozen sample manifest with a fixed random seed.

Benchmark	Train	Validation	Test	Total
BBH	2619	266	190	3075
ARC-Challenge	779	200	150	1129
OpenBookQA	623	200	150	973
GSM8K	2922	500	250	3672
MATH500	350	50	100	500
Total	**7293**	**1216**	**840**	**9349**

C.3 Prompt Templates

Below we describe the prompt templates used for each benchmark family. Demonstrations are fixed across all samples within a benchmark. All five training benchmarks use direct-answer prompts to encourage concise responses during exhaustive oracle collection. The short-budget protocol is used for oracle collection and the ablations described in Section 4. For the main results in Table 1, generation proceeds until the model completes its response, without an imposed generation-length limit. Thus, the direct-answer instruction specifies the requested response style; the generation limit is a separate evaluation setting.

ARC-Challenge and OpenBookQA. Three fixed demonstrations precede the test question. Each demonstration and the test item follow the format:

```
Q: <question text>
Options:
(A) <option 1>
(B) <option 2>
(C) <option 3>
(D) <option 4>
A: (<letter>)
```

The model generates from `A:` onward. ARC-Challenge and OpenBookQA each use their own set of three demonstrations.

GSM8K. A system instruction requests that the model output only the final number without showing any explicit reasoning. Two demonstrations follow the instruction, each ending with the bare numerical answer. The test item is formatted as:

```
<instruction: output only the final number>

Q: <demo 1>
A: The answer (arabic numerals) is <number>

Q: <demo 2>
A: The answer (arabic numerals) is <number>

Q: <test question>
A: The answer (arabic numerals) is
```

The scorer extracts the last number from the output and compares it numerically to the reference ($|\text{pred} - \text{gold}| < 10^{-3}$).

MATH500. The same structure as GSM8K, with the instruction requesting a `\boxed{}` answer. Two demonstrations include boxed answers (e.g., `\boxed{2}`, `\boxed{\frac{1}{2}}`). The scorer extracts the last `\boxed{}` expression and normalizes it using standard rules (fraction simplification, percentage conversion, equation right-hand-side extraction) before string comparison.

BBH. Each subtask uses its official BIG-Bench-Hard task description and three demonstration questions. The demonstrations are derived from the official chain-of-thought prompts by stripping the reasoning text and retaining only the final answer letter:

```
<official task description>

Q: <demo question 1>
A: (<letter>)

Q: <demo question 2>
A: (<letter>)

Q: <demo question 3>
A: (<letter>)

Q: <test question>
A:
```

The scorer extracts the first answer letter or parses the output for affirmative or negative keywords using a priority cascade: parenthesized single letter, "the answer is" phrase, then earliest keyword match..

HellaSwag, TruthfulQA-MC1, GPQA-Diamond. All use 0-shot multiple-choice with a direct-answer instruction and no demonstrations:

```
Q: <question or sentence prefix>
Options:
(A) <candidate 1>
...
Answer with only the option letter.
A:
```

Option order is deterministically shuffled per item to remove positional bias. TruthfulQA-MC1 has a variable number of candidates (2 to 13).

2WikiMultiHop (extractive QA). The model receives a set of Wikipedia paragraphs and a multi-hop question, and must produce a answer:

```
<instruction: answer with a short phrase>

Context:
[Title 1] <paragraph text>
[Title 2] <paragraph text>
...

Question: <multi-hop question>
Answer:
```

The scorer first attempts exact match (after lowercasing and stripping articles/punctuation), then falls back to token-level F1.

C.4 Collection Process

For each of the 9349 samples and each of the 25 actions in $\mathcal{A}$, we execute a single greedy generation on the frozen backbone and record the sample identifier, action identifier, binary score, analytic cost, utility, and the generated text. This produces 233,725 records per backbone.

During the BASE forward pass of each sample, we additionally record the depth-wise hidden trajectory used as the selector's input feature. At $L{=}10$ pre-selected layers (spaced to cover the mid-to-late portion of each backbone), we store the last-token hidden state, yielding a tensor of shape $[10, d_h]$ per sample. The trajectory is collected once per sample, during the BASE pass only. Non-BASE actions do not record trajectories, because the selector conditions on the model's computation before any morphology is applied.

Collection is parallelized across GPUs by sharding over samples (8 shards for the 27B backbone, 16 for the 4B and 12B backbones). Each record is flushed to disk immediately, and collection supports interruption and exact resumption at the granularity of individual (sample, action) pairs.

C.5 Baselines

Four fixed baselines are fitted on the train split per backbone. The fitting criterion in every case is the action with the highest mean utility on the relevant subset of training samples.

- **Base LM**: always selects BASE.
- **Fixed-REPEAT**: the single REPEAT action with the highest mean utility across all training samples.
- **Fixed-BRANCH**: the single BRANCH action with the highest mean utility across all training samples.
- **Bench Prior**: one action per benchmark unit, each chosen as the highest-mean-utility action on that unit's training samples. Different units may select different actions, but all samples within a unit share the same action.

D Selector Architecture and Training

This appendix specifies the selector network and training procedure for each backbone. The Method section describes the selector functionally (Section 3.2). Here we give the concrete dimensions, parameter counts, and optimization settings.

D.1 Input Representation

The selector receives a depth-wise hidden trajectory $H \in \mathbb{R}^{L \times d_h}$ extracted during the BASE forward pass, with $L{=}10$ for all three backbones. The 10 layers are chosen to span the mid-to-late portion of each backbone: layers 20, 22, ..., 38 for Qwen3.5-27B, layers 12, 14, ..., 30 for Qwen3.5-4B, and layers 4, 8, ..., 40 for Gemma4-12B.

Before entering the network, each layer slice H_l is passed through RMSNorm ($\epsilon{=}10^{-6}$) and scaled elementwise by γ_l, the pretrained gain vector of the backbone's own input layer normalization at the consuming layer. These gains are frozen buffers, not trainable parameters. The normalization aligns the selector's input with the numerical semantics the backbone uses at each depth.

D.2 Network Architecture

Figure 6 illustrates the selector's three-stage architecture. We describe each stage below with the computation it performs.

(a) Encoder. Each layer slice $H_l \in \mathbb{R}^{d_h}$ first passes through the pre-normalization described in the preceding subsection. A single shared linear projection maps the result to a d-dimensional feature:

$$s_l = W_{\text{proj}}\,\text{PreNorm}(H_l), \qquad W_{\text{proj}} \in \mathbb{R}^{d \times d_h}, \quad l = 1, \ldots, L.$$

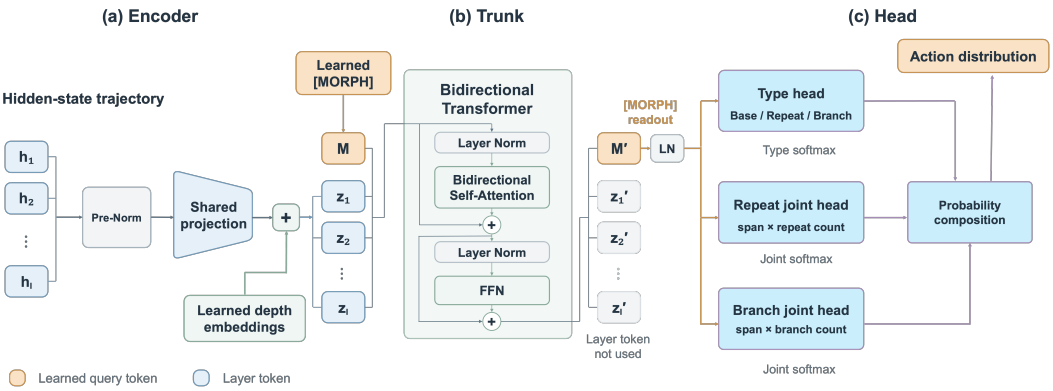


Figure 6: Selector architecture. (a) Encoder: per-layer hidden states are pre-normalized and projected to a shared low-dimensional space, then combined with learned depth embeddings. (b) Trunk: a learned query token [MORPH] and the L layer tokens attend to one another in a bidirectional transformer block. The output at the query position serves as the decision vector. (c) Head: the decision vector is mapped to a type distribution and two joint span-intensity distributions, composed into the final action distribution.

On Gemma4-12B, the encoder additionally computes a first-order difference $\delta_l = s_l - s_{l-1}$ (with $\delta_1 = \mathbf{0}$) and concatenates it with s_l, producing a $2d$-dimensional feature per layer.

A fusion layer projects each feature to the trunk width d_z and adds a learned depth embedding:

$$z_l = W_{\text{fuse}}\, s_l + e_l, \qquad W_{\text{fuse}} \in \mathbb{R}^{d_z \times d}, \quad e_l \in \mathbb{R}^{d_z},$$

where $\{e_1, \ldots, e_L\}$ is a learned embedding table of size $L \times d_z$, the only source of positional information in the network.

(b) Trunk. A learned query token $\mathbf{M} = [\text{MORPH}] \in \mathbb{R}^{d_z}$ is prepended to the sequence of L layer tokens, forming a length-$(L{+}1)$ input:

$$\mathbf{Z}^{(0)} = [\,\mathbf{M},\ z_1,\ z_2,\ \ldots,\ z_L\,] \in \mathbb{R}^{(L+1)\times d_z}.$$

This sequence is processed by a single pre-norm bidirectional transformer block with n_h attention heads and an FFN of expansion factor $f{=}4$. All $L{+}1$ tokens attend to one another without a causal mask:

$$\hat{\mathbf{Z}} = \mathbf{Z}^{(0)} + \text{BiAttn}\big(\text{LN}(\mathbf{Z}^{(0)})\big), \tag{8}$$

$$\mathbf{Z}^{(1)} = \hat{\mathbf{Z}} + \text{FFN}\big(\text{LN}(\hat{\mathbf{Z}})\big). \tag{9}$$

The output at position 0 (the query token), passed through a final LayerNorm, serves as the decision vector:

$$z_{\text{out}} = \text{LN}\big(\mathbf{Z}_0^{(1)}\big) \in \mathbb{R}^{d_z}.$$

The remaining L layer-token outputs are discarded. Information from each depth reaches z_{out} through bidirectional attention, which acts as a learned, content-adaptive pooling over the trajectory.

(c) Head. The head maps z_{out} to a distribution over $\mathcal{A}$ through three parallel branches.

Type head. A linear layer produces logits over $\{\text{BASE}, \text{REPEAT}, \text{BRANCH}\}$:

$$\ell_\tau = W_\tau\, z_{\text{out}} + b_\tau \in \mathbb{R}^3, \qquad p_\phi(\tau \mid H) = \text{softmax}(\ell_\tau).$$

Repeat joint head. A two-layer MLP produces $S \times P$ joint logits over span and intensity:

$$\ell_{\text{REPEAT}} = W_2^{\text{REPEAT}}\, \text{GELU}\big(W_1^{\text{REPEAT}}\, z_{\text{out}}\big) \in \mathbb{R}^{S \cdot P},$$

reshaped to $[S, P]$ and passed through a single softmax over all $S{\cdot}P$ entries: $p_\phi(s, \theta \mid \text{REPEAT}, H) = \text{softmax}(\ell_{\text{REPEAT}})_{s,\theta}$.

Branch joint head. Same structure as *Repeat joint head*.

Probability composition. The final action probability is:

$$p_\phi(m \mid H) = \begin{cases} p_\phi(\textsc{Base} \mid H), & m = \textsc{Base}, \\ p_\phi(\tau \mid H)\, p_\phi(s, \theta \mid \tau, H), & m = (\tau, s, \theta). \end{cases}$$

For all three backbones $S{=}4$ spans and $P{=}3$ intensity levels.

D.3 Dimensions and Parameter Counts

Table 8 lists the concrete dimensions. The selector is small relative to the backbone.

Table 8: Selector dimensions and parameter counts. All parameters are trainable. The pre-norm gains (γ_l) are frozen buffers and not included.

	Qwen3.5-27B	Qwen3.5-4B	Gemma4-12B
Backbone d_h	5120	2560	3840
Encoder output d	192	192	384
Trunk input	d	d	$2d$
Trunk width d_z	64	64	128
Attention heads n_h	2	2	4
Blocks n_{blk}	1	1	1
Head hidden d_{head}	48	48	96
Encoder params	983,232	491,712	1,474,944
Trunk params	63,168	63,168	298,368
Head params	7,611	7,611	27,483
Total per selector	**1,054,011**	**562,491**	**1,800,795**

D.4 Training

Table 9 lists the optimization settings. Training minimizes the distillation loss in Equation 6 against the teacher distribution $q_T(m \mid x)$.

Table 9: Training hyperparameters. Values that differ across backbones are separated by slashes (27B / 4B / Gemma).

Optimizer	AdamW
Peak learning rate	8×10^{-4} / 8×10^{-4} / 7×10^{-4}
Schedule	Cosine decay, 5% linear warmup, min = 2% of peak
Gradient clipping	Global norm 5.0
Effective batch / micro-batch	256 / 128 (accumulation = 2)
Max epochs	50 / 50 / 50
Early stopping patience (epochs)	8 / 8 / 8
Early stopping metric	Mean utility on validation set
Improvement threshold	10^{-5}
Distillation temperature T	0.15 / 0.01 / 0.5
Cost coefficient λ	0.05
Head dropout	0.05 / 0.05 / 0.1

Early stopping monitors mean utility on the validation split, using the highest-probability action predicted by the selector for each input. The checkpoint with the highest validation mean utility is retained.

E Inference Optimization and Cost Measurement

A direct implementation of SelfMorph evaluates overlapping prompt prefixes during probing and execution, and stores identical cache rows outside the branch span. We eliminate these redundancies through prefix reuse and lossless cache deduplication, respectively. The optimizations preserve the selected morphology, model weights, precision, perturbations, and decoding rule. This appendix describes their equivalence conditions and examines the resulting time and memory costs.

E.1 Reusing Computation Across Probe and Commit

Truncated probing and prefix reuse. Let $\mathcal{I}$ be the observed layer indices and $L = 1 + \max \mathcal{I}$. The probe executes layers $[0, L_{\text{probe}})$ and collects the requested last-token states during prefill. Layers beyond $L_{\text{probe}} - 1$ cannot affect the trajectory and are omitted. We retain the full prompt hidden states and caches at reusable boundaries.

For Base, execution resumes at depth L_{probe}, so probe and commit together perform one ordinary prefill. For Repeat, the layer sequence is

$$[0, j) \ + \ [i, j) \times r \ + \ [j, N).$$

The first j evaluations are shared with the ordinary forward pass. We therefore reuse the first $\min(L_{\text{probe}}, j)$ evaluations, then continue along the selected sequence. Repeated layer occurrences maintain separate cache slots, and mutable states are copied before reuse to protect the saved prefix. Across probe and commit, the prompt-level layer counts are

$$E_{\text{Base}} = N, \qquad E_{\text{REP}} = N + r(j - i) + \max(L_{\text{probe}} - j, 0).$$

The last term accounts for probe evaluations beyond the point where Repeat diverges from the ordinary path. Reuse preserves the inputs, layer order, and cache states of the shared computation. Exact trajectory comparisons and dedicated checks of 13 prefix-compatible actions on two prompts confirm unchanged states and generated tokens in our evaluation environment.

Preserving branch execution. For a single input, the reference Branch implementation executes the stack at batch size K, while branching and merging hidden states within the selected span. Collapsing the batch outside that span changes execution shapes and can change floating-point accumulation through kernel selection. With identical perturbations, this alternative changed 58 of 144 reference outputs in a controlled comparison. We retain the original batch shape and execute a separate branch prefill. The next optimization removes duplicate storage while preserving that execution shape.

E.2 Lossless Deduplication of Branch Caches

A cache invariant. For standard-attention layers outside the branch span, replica cache rows are identical after prefill: the replicas process the same inputs before branching and the same merged states after branching. Each subsequent decoding step appends identical keys and values at these layers, preserving equality of the complete cache histories. We exploit this invariant by storing one row and exposing K logical rows through a broadcast view with zero stride along the replica dimension. Layers inside the span retain their branch-specific histories.

Updates append keys and values to the single stored row before broadcasting the result for attention. This avoids materializing repeated histories during concatenation. Sliding-window truncation and cache positions follow the original update rules. Attention receives the same logical key and value tensors as in the reference execution. Under the evaluated backend, direct comparisons for $K \in \{2, 4, 8\}$ confirm bit-identical attention outputs. End-to-end checks also produce identical text on three prompts per configuration for Gemma with span $(0, 10)$ and Qwen3.5-4B with span $(3, 8)$ at $K = 4$.

Storage follows the distinct histories. Let b_ℓ be the single-replica cache size of layer ℓ at a fixed sequence length, and let $\mathcal{D}$ contain the layers eligible for deduplication. Define ρ as the fraction of single-replica cache bytes belonging to layers that retain separate copies:

$$\rho = \frac{\sum_{\ell \notin \mathcal{D}} b_\ell}{\sum_\ell b_\ell}.$$

The retained cache footprint is then

$$B_{\text{dedup}} = \sum_{\ell \in \mathcal{D}} b_\ell + K \sum_{\ell \notin \mathcal{D}} b_\ell, \qquad \frac{B_{\text{dedup}}}{B_{\text{single}}} = 1 + (K-1)\rho,$$

where $B_{\text{single}} = \sum_\ell b_\ell$. With uniform standard-attention caches, only the span retains multiple histories, so $\rho = (j-i)/N$. The byte-weighted expression also accommodates sliding-window and hybrid architectures.

The Qwen backbones additionally maintain linear-attention recurrent and convolutional states. Their current interfaces update fixed-address buffers in place, which does not support writes to aliased broadcast rows. These states remain replicated and contribute to ρ. Consequently, savings depend on the fraction of cache bytes eligible for sharing, rather than the number of eligible layers alone. In a controlled Gemma $K = 4$ comparison, deduplication removes 58.5% of retained cache storage at approximately 1.8% additional decode time. Both the storage reduction and the update overhead are included in the cost measurements below.

E.3 Cost Evaluation

Analytic cost. We summarize the cost trade-off between morphologies using a static cost function. This provides a consistent training penalty without introducing workload variation and measurement noise from individual wall-clock observations:

$$C(m) = 1 + \frac{j-i}{N} k_{\text{eff}}(\tau, \theta), \qquad k_{\text{eff}}(\tau, \theta) = \begin{cases} 0, & \tau = \text{BASE}, \\ r, & \tau = \text{REPEAT}, \\ \kappa_K - 1, & \tau = \text{BRANCH}. \end{cases}$$

For REPEAT, the intensity coefficient follows the number of additional passes. For BRANCH, we use $(\kappa_2, \kappa_4, \kappa_8) = (1.3, 1.6, 2.6)$ as effective penalty multipliers, informed by execution time and cache measurements of the optimized implementation. Parallel execution can accommodate additional branches with little change in decode throughput, while retaining distinct branch histories increases storage requirements. The multipliers incorporate both considerations into the training penalty. Together with the span-width factor, they define $C(m)$; subtracting one sets the penalty for BASE to zero. All coefficients remain fixed during selector training.

Measurement protocol. We measure all 840 held-out inputs on each backbone using `bfloat16` weights, PyTorch 2.13.0+cu130, Transformers 5.15.1, and framework attention implementations. Deduplication is enabled for every strategy that selects BRANCH.

Figures 7 and 8 measure morphology execution. For each strategy, the generation-call interval covers prompt prefill and response decoding under its selected action.

Each input–strategy pair receives one warmup and two timed runs, whose times are averaged. Strategy order rotates across inputs, and allocator caches are retained between runs. Synchronization separates prefill from decoding, with no per-token synchronization during decoding. Let t_{gen} be the generation-call time and t_{prefill} the first forward-pass time. For n generated tokens, the reported rates are

$$q_{\text{decode}} = \frac{n-1}{t_{\text{gen}} - t_{\text{prefill}}}, \qquad q_{\text{gen}} = \frac{n}{t_{\text{gen}}}.$$

The first token is produced by prefill and excluded from the decode numerator. Bars show sample-mean rates; annotations compare these means with the corresponding BASE mean.

Memory accounting. Resident memory is sampled after each forward pass and averaged first within a generation call, then across inputs. Subtracting frozen weights gives runtime overhead, including caches and other runtime allocations. Cache bytes count distinct backing allocations once, including when several logical rows share storage. The figure reports average resident memory; temporary peaks are a separate quantity.

Structural predictions and measured cost behavior. For REPEAT, the analytic model makes a directly testable prediction: adding $r(j-i)$ layer evaluations increases the executed-depth ratio to

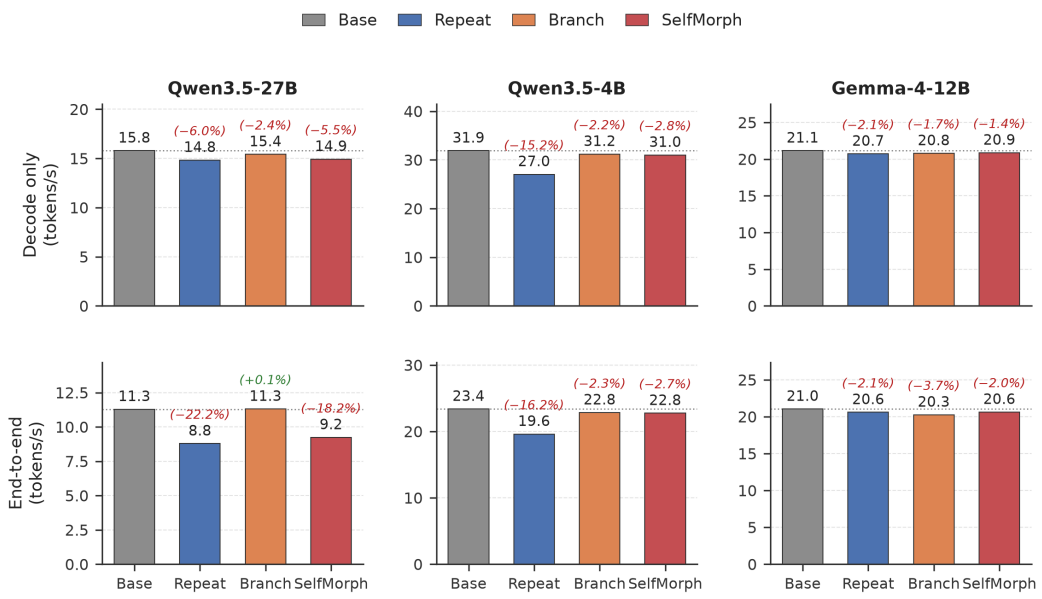


Figure 7: Generation throughput on 840 inputs per backbone, with branch-cache deduplication enabled. Top: decode-phase rate, excluding prefill and its first output token. Bottom: generation-call throughput, including prompt prefill. Bars show sample means; annotations compare each mean with BASE. Strategies can produce different output lengths, so throughput and per-query latency need not have the same ordering.

$C(m)$, implying a decode-rate ratio near $1/C(m)$ when layer costs are comparable. Across the three fixed REPEAT configurations, the measured decode-rate ratios differ from this prediction by less than 0.7 percentage points. Measured cache ratios also agree with $C(m)$ to within 0.8% relative error. Thus, both execution time and storage track the structural depth term across the three backbones.

For BRANCH, the measurements reveal distinct effects on decoding speed and storage. Across the three fixed branch configurations, decode throughput remains within 2.4% of BASE (Figure 7), consistent with amortized weight access during parallel execution. Additional branches retain distinct histories, increasing cache storage, while prefill cost depends on the expanded batch and prompt length. These complementary resource effects motivate considering both execution time and cache footprint when setting the proxy coefficients.

The cache-sharing model predicts the storage increase independently: for Gemma's span $(0, 10)$ at $K = 4$, uniform per-layer caches give $1 + 3(10/48) = 1.625$ times single-replica storage, close to the measured ratio of approximately 1.661. The byte-weighted formulation further accounts for unequal layer capacities and the recurrent states retained by hybrid backbones.

The Qwen strategies produce different output lengths under the same limits. Their generation-call rates therefore reflect both execution speed and the number of tokens over which prefill is amortized: a shorter answer can finish sooner yet have lower tokens/s. Gemma has matched output lengths in this workload, making the relationship between its two timing panels more direct.

Figure 8 isolates runtime overhead because frozen weights dominate total memory. Gemma benefits most from branch-cache sharing, consistent with its standard-attention caches. The hybrid Qwen models retain replicated recurrent states, reducing the share of storage that deduplication can remove. Together, the measurements show how serial depth, parallel execution, and cache sharing contribute differently to the cost of each morphology.

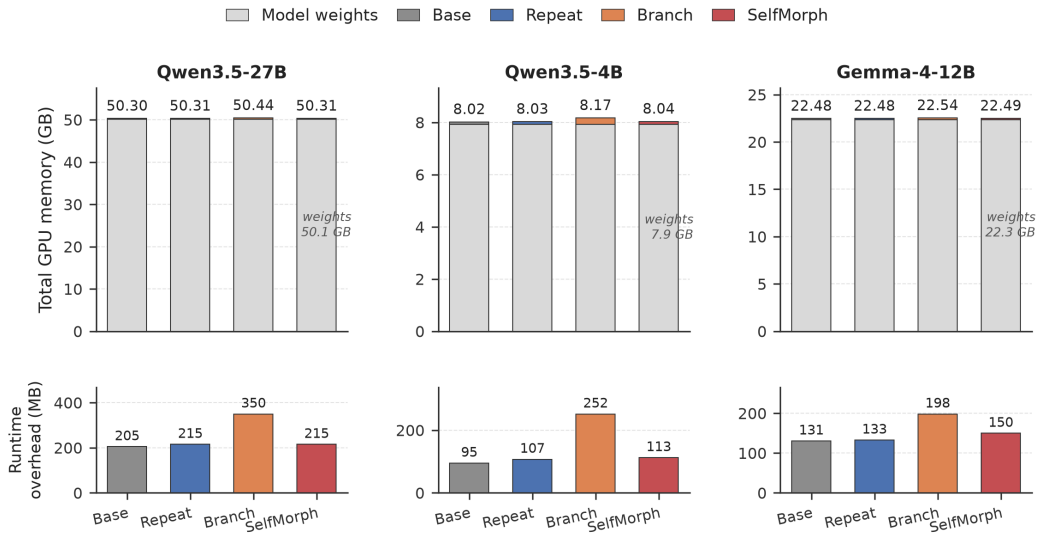


Figure 8: Average resident GPU memory during generation on the same 840 inputs. Top: frozen weights and runtime overhead. Bottom: runtime overhead on a separate scale. Eligible branch caches share physical storage; recurrent states with incompatible in-place updates remain replicated. Values are resident averages rather than peaks.

F Morphology Analysis

This section presents two complementary views of how the morphology types behave on frozen backbones: the geometric signature each type leaves on the hidden state trajectory, and the task-level dissociation between Repeat and Branch.

F.1 Trajectory Geometry

Figure 9 illustrates the effect of each morphology type on the hidden state trajectory of a frozen Qwen3.5-27B backbone, using an ARC-Challenge question.

The three morphology types produce qualitatively distinct trajectory signatures. Base (gray) follows the standard depth profile. Repeat (red) extends the trajectory past the depth the base forward ever reaches: with $r{=}2$, the representation passes through the span three times, with each traversal receiving the state produced by the preceding traversal. Branch (green) fans out at the span entrance into K diverging trajectories and collapses them into a single merged state at the exit. The merged state resumes the standard depth profile from that point, but at a different position in hidden space than the base trajectory would have reached.

F.2 Task-Level Dissociation

The two non-Base families help on different inputs. For every training sample, we records the utility of all 25 actions. If a single fixed action were always optimal, per-sample selection would be unnecessary. The oracle distribution shows it is not.

Figure 10 plots, for each task group, the net effect of each backbone's single best Repeat action (x-axis) against that of its single best Branch action (y-axis). Points in the shaded off-diagonal quadrants indicate tasks where one morphology actively hurts while the other helps. On Qwen3.5-27B, some BBH subtasks improve under Repeat but decline under Branch, whereas many science sub-blocks improve under both. Opposite-sign effects also occur on the smaller backbones, although their magnitudes and task-group distributions differ. These patterns show that the two fixed interventions have distinct task-group effects.

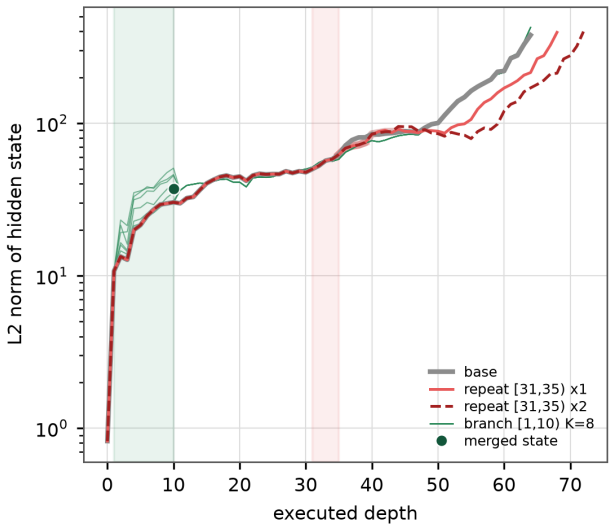


Figure 9: Hidden state L2 norm (log scale) across executed depth for one ARC-Challenge context (frozen Qwen3.5-27B). Gray: unmodified BASE trajectory. Red solid/dashed: REPEAT with $r{=}1$ and $r{=}2$ extra passes through span $[31, 35)$ (shaded red). Green: BRANCH with $K{=}8$ replicas through span $[1, 10)$ (shaded green). The eight thin lines fan out from the entry and collapse into one merged state (dark dot) at the exit.

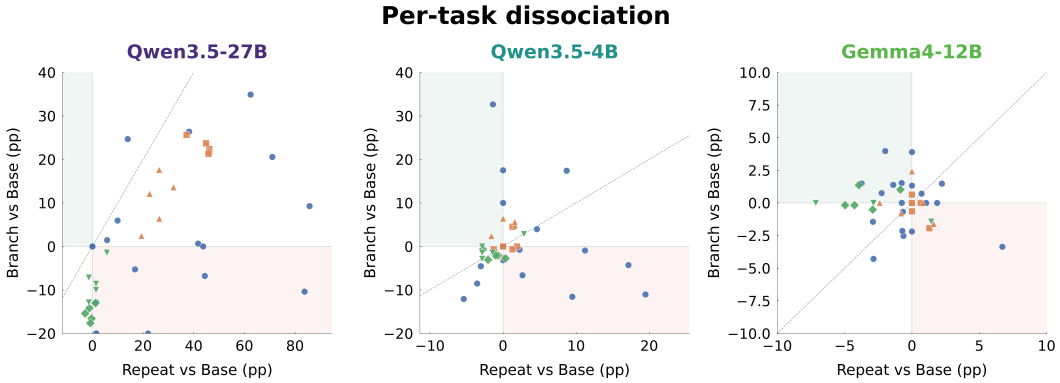


Figure 10: Per-task dissociation between REPEAT and BRANCH on the training split. Each point is one BBH subtask (○) or one of five sample sub-blocks of ARC-Challenge (□), OpenBookQA (△), GSM8K (◇), or MATH-500 (▽), colored by task domain (blue = reasoning, orange = science MC, green = mathematics). Axes show the net effect (help% − hurt%) relative to BASE. Points in the shaded off-diagonal quadrants indicate that one morphology actively hurts while the other helps on the same task group.

G ADDITIONAL ABLATION STUDIES

We complement Section 4.3 with ablations of selector architecture, capacity, distillation temperature, and cost coefficient. Unless otherwise noted, ablations are conducted on Qwen3.5-27B under the short-budget protocol, with all other hyperparameters held fixed. Mean score denotes the average per-sample task score.

Architecture components. Table 10 reports ablation results for removing two selector components. Self-attention lets the query token aggregate cross-layer trajectory information, weighting each depth by input-dependent relevance so that highly informative layers shape the final decision vector. Pre-normalization applies the backbone's corresponding normalization to the trajectory features.

Table 10: Architecture ablation. Δ is the absolute difference in mean score from the default architecture.

Variant	Mean Score	Δ
Default	**0.6083**	
w/o PreNorm	0.6036	-0.0047
w/o Attention	0.6012	-0.0071

Table 11: Capacity ablation on Qwen3.5-27B. Bold marks the deployed configuration.

d	d_z	d_{head}	Params	Mean Score
96	48	24	528K	0.6036
192	**64**	**48**	**1.05M**	**0.6083**
384	128	96	2.24M	0.6095
512	192	128	3.22M	0.6060
1024	256	256	6.44M	0.6024

Table 12: Temperature ablation on Qwen3.5-27B. Bold marks the deployed setting.

T	Mean Score
0.01	0.6024
0.10	0.6071
0.15	**0.6083**
0.20	0.6036
0.40	0.5964

Removing either component lowers the observed mean score, supporting their inclusion in the selector.

Capacity. Table 11 reports the effect of scaling the selector across a 12$\times$ parameter range. Among the evaluated configurations, the highest score occurs at 2.24M parameters. The deployed 1.05M setting performs close to this configuration with fewer parameters. Both the smallest configuration and those above 3M parameters obtain lower scores

Temperature. Table 12 varies the temperature T of the teacher distribution $q_T(m \mid x)$. Among the tested values, T=0.15 achieves the highest score. As described in Section 3.1, lower temperatures concentrate the teacher on higher-utility actions, while higher temperatures increase its entropy and reduce probability differences between actions. Exact utility ties retain equal probability at any temperature. The results favor an intermediate temperature for this backbone, balancing concentrated utility preferences with a broader distribution over alternative actions.

Cost coefficient λ. Figure 11 shows the effect of the cost coefficient λ in the utility $U(x, m) = \text{score}(x, m) - \lambda(C(m) - 1)$. Mean analytic cost generally decreases as λ increases. Increasing λ strengthens the teacher's preference for lower-cost actions. A small positive coefficient improves mean score over $\lambda = 0$ on all three backbones, while larger coefficients generally reduce score. For actions with equal task scores, the cost term favors the less expensive alternative. The observed gains suggest that this preference can also benefit selection in the evaluated setting.

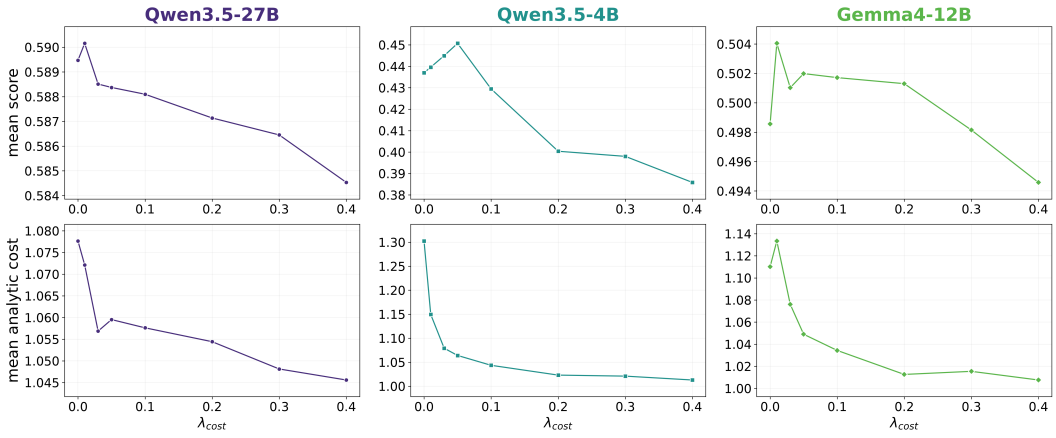


Figure 11: Sensitivity to the cost coefficient λ across three backbones on the training split. Top: mean score. Bottom: mean analytic cost. Each point represents a separately trained SELFMORPH.