NEST: TRAINING CONTINUOUS LATENT REASONING AS ANYTIME COMPUTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Adaptive latent reasoning aims to allocate computation according to input difficulty, but early stopping is only useful when intermediate latent states are themselves reliable answer endpoints. We study this mismatch and propose NEST (Nested Endpoint Supervision for Trajectories), an anytime-training framework that trains variable-depth competence before learning variable-depth execution. NEST supervises multiple prefixes of a single shared recurrent latent trajectory to produce complete answers, so each intermediate state serves both as a valid prediction endpoint and as the continuation state for deeper reasoning. The full method further uses decision distillation, ordered alignment, and monotonic refinement to structure this hierarchy, followed by a calibrated student-only policy for adaptive execution. Across eight reasoning benchmarks with Qwen3.5-4B, NEST achieves 82.68 average accuracy at 0.402 relative GPU cost, compared with 82.07 accuracy at 0.592 cost for SwiReasoning. Post-hoc comparisons on the test frontier obtained after all calibration-selected thresholds are frozen show 38.3% lower GPU compute at SwiReasoning-level accuracy and a 0.95-point gain at the same reported GPU cost; these matched readouts are descriptive comparisons, not threshold-selection targets. Fixed-prefix and matched-router controls show that the gains arise from improved shallow-prefix competence rather than stronger halting machinery. Adding Core NEST to SIM-CoT and CODI similarly improves accuracy while reducing latent depth, and pool-free deployment with disjoint calibration preserves the operating regime. These results support nested endpoint supervision as an effective strategy for adaptive continuous latent reasoning.

1 INTRODUCTION

Continuous latent reasoning moves part of the reasoning process from natural-language tokens into recurrent hidden states, reducing textual overhead and making input-dependent computation possible. Coconut recurrently feeds hidden states back as continuous thoughts, while CODI distills explicit chain-of-thought into continuous representations (Hao et al., 2024; Shen et al., 2025). These methods demonstrate that latent trajectories can support strong final-answer prediction, but adaptive execution imposes an additional requirement: an intermediate state may be useful for continuing computation without yet being a reliable state from which to answer.

Existing continuous latent reasoning methods typically optimize intermediate latent states as part of a computation trajectory toward a final answer. Although these states receive end-to-end training signals, they are not necessarily trained to serve as independently answer-capable endpoints. We refer to the resulting mismatch as an *endpoint-supervision gap*: under adaptive computation, inference may terminate at an intermediate latent state whose training role was primarily to support subsequent computation rather than to produce a complete prediction. Consequently, learning when to stop is insufficient if the candidate stopping points themselves have not been explicitly trained to be reliable endpoints. We therefore study adaptive latent reasoning through an *anytime computation* formulation with two coupled requirements: make intermediate computation states answer-capable, then determine when sufficient computation has been reached.

Figure 1 visualizes the distinction between terminal-only training and NEST's shared recurrent endpoint supervision.

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.

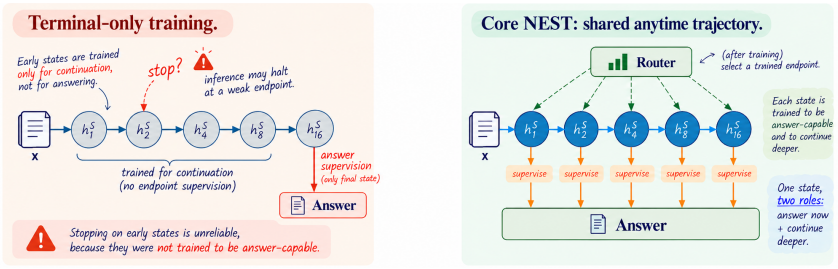


Figure 1: Why adaptive stopping requires endpoint supervision. Under terminal-only training, intermediate recurrent states participate in the shared computation trajectory but are not explicitly trained to serve as answer-capable stopping points. Core NEST supervises multiple prefixes of the same recurrent trajectory as valid answer endpoints. Each student state therefore has two roles: it can answer at its current depth and remains the exact continuation state from which deeper computation proceeds. Adaptive execution is fitted only after this anytime competence has been learned.

We introduce NEST (Nested Endpoint Supervision for Trajectories). Its intellectual center is deliberately simple: **nested endpoint supervision on a shared recurrent continuation**. A depth-K state must support a complete answer now and simultaneously remain the exact state prefix from which every deeper endpoint continues. We call direct multi-depth answer training *Core NEST*. Decision distillation, ordered teacher-state alignment, and monotonic refinement are structure-enhancing components that improve the hierarchy after this core competence has been created; a calibrated student-side execution policy then decides whether to accept, continue the cached trajectory, or fall back to explicit reasoning. This ordering separates *endpoint quality* from *allocation quality*.

Why adaptive stopping changes the training target. A terminal latent reasoner only needs its final state to support answer generation; earlier states may remain useful purely because they help produce the next recurrent state. An adaptive system imposes a stricter requirement: every state at which execution may stop must itself be a legitimate prediction point. This distinction matters because a stopping policy cannot repair a weak endpoint after it has decided to halt. NEST therefore treats endpoint competence as a property to be learned before routing is fitted, rather than expecting confidence estimation alone to discover reliable early exits. The main experiments mirror this decomposition: fixed-prefix and matched-router studies evaluate whether the states themselves become usable, while frontier and calibration studies evaluate whether those states can then be allocated efficiently.

Primary contributions. Our primary contributions are:

- **Anytime-training formulation.** We formulate a practical endpoint-supervision gap for adaptive continuous latent reasoning and study the design rule *train variable-depth competence before variable-depth execution*. Core NEST implements this rule by supervising multiple answer endpoints on one shared recurrent trajectory.
- **Evidence for shallow-prefix competence.** Across all eight benchmarks and every tested depth $K \in \{1, 2, 4, 8, 16\}$, Core NEST and full NEST improve shallow-prefix accuracy over terminal-only training, while the three approaches nearly converge at $K = 16$. A matched-router control further separates endpoint training from halting-policy capacity. NEST primarily makes the trajectory *anytime usable*; it does not mainly improve the terminal reasoner.
- **A stronger accuracy–computation frontier.** Across eight benchmarks, NEST reaches 82.68 average accuracy versus 82.07 for SwiReasoning while reducing Rel.GPU from 0.592 to 0.402. Under a matched data/teacher/compute control it retains a $+0.33$-point accuracy advantage with 28.8% lower GPU cost. After the calibration-selected thresholds are frozen and evaluated on test, a post-hoc frontier comparison gives 38.3% lower compute at SwiReasoning-level accuracy.
- **Transfer and deployment controls.** Adding Core NEST to SIM-CoT and CODI improves eight-benchmark accuracy while reducing latent depth under the same execution policy. Pool-free inference and strictly disjoint calibration preserve the operating regime without accessing training-time candidate artifacts.

Code. An anonymous implementation and reproduction package is available at https://doi.org/10.5281/zenodo.22894522.

2 Related Work and Background

Continuous latent reasoning moves intermediate computation from explicit text into hidden representation space. Coconut recurrently feeds continuous hidden states (Hao et al., 2024; Goyal et al., 2024); CODI trains explicit-to-implicit transfer through self-distillation (Hinton et al., 2015; Shen et al., 2025); SIM-CoT studies step-level supervision for longer implicit trajectories (Wei et al., 2026). More broadly, CoT, process supervision, and rationale-learning methods show that intermediate computation can provide training signals unavailable from final outcomes alone (Wei et al., 2022; Wang et al., 2023; Zhou et al., 2023; Uesato et al., 2022; Lightman et al., 2024; Zelikman et al., 2022; 2024). NEST makes the location of task competence along one recurrent latent trajectory an explicit training target by requiring several computation prefixes to support complete answers.

Adaptive computation studies how input-dependent inference effort can be reduced through intermediate exits, recurrent halting, or dynamic depth (Lee et al., 2015; Teerapittayanon et al., 2016; Graves, 2016; Dehghani et al., 2019; Banino et al., 2021). Recent language-model systems extend this idea to confidence-based computation and latent reasoning (Schuster et al., 2022; Wang et al., 2025; Shi et al., 2026). Deep supervision and early exiting are direct conceptual antecedents, but the supervised object is different in NEST: a depth-K endpoint is the same recurrent state prefix that generates every deeper endpoint, rather than an auxiliary branch attached to an intermediate feed-forward layer. The shallow state must therefore be answer-capable now and continuation-capable later. NEST separates endpoint quality from allocation quality: fixed-depth and representation diagnostics test whether an intermediate latent prefix is competent, while learned, random, and oracle policies test whether a given input is assigned an appropriate amount of computation.

Relation to deep supervision and early exiting. Deep supervision is the closest analogy, but recurrence changes the coupling. In a feed-forward early-exit network, an auxiliary classifier can be a side readout from an intermediate representation. In NEST, the supervised prefix is also the state reused by every deeper endpoint. Improving a shallow answer while damaging continuation would therefore hurt deeper computation. This shared-state constraint motivates treating Core NEST as anytime-trajectory training rather than independent exit heads.

3 NEST: Nested Endpoint Supervision for Trajectories

NEST has one central design: train a *single recurrent latent trajectory* so that several prefixes are already valid answer endpoints before learning when to stop. The explicit reasoning teacher is used only to provide training-side structure; the recurrent student is the model that is optimized and deployed. Throughout, superscripts S and T denote student and frozen-teacher quantities, respectively. Figure 2 mirrors the three method subsections below.

3.1 Training objects and staged design

For each hierarchy-training example, the student receives question x and gold answer y. A frozen teacher, prompted with the question only, produces an explicit trace $\rho^{\mathrm{T}} = (\rho_1^{\mathrm{T}}, \ldots, \rho_R^{\mathrm{T}})$; a detached teacher representation $\mathbf{h}_s^{\mathrm{T}}$ is extracted for each reasoning step. The gold answer is used for supervised endpoint CE, while teacher traces and candidate scores provide auxiliary training structure. No teacher quantity is available at deployed inference.

Training is intentionally staged. Stage A constructs these training-only artifacts. Stage B trains the recurrent hierarchy. Stage C freezes the hierarchy and fits the capacity/confidence modules on a dedicated router-fitting split; a disjoint calibration split selects the final acceptance threshold. The stages are not one joint optimization problem.

Separation of responsibilities. Stage B asks whether each prefix is competent without using a learned halting decision; Stage C allocates computation on the frozen hierarchy. Freezing prevents router fitting from changing candidate endpoint competence, while the separate calibration split keeps

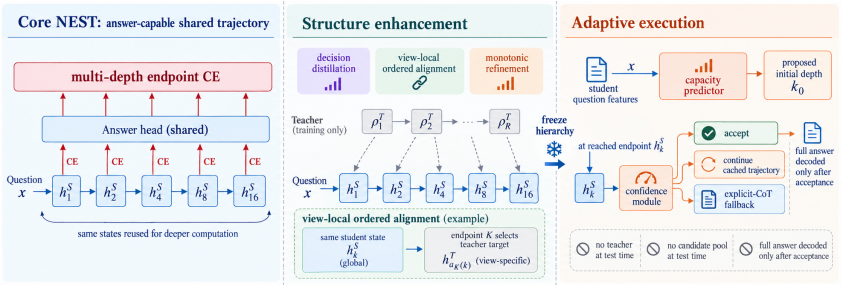


Figure 2: Overview of NEST. The method separates endpoint competence from computation allocation. Core NEST first trains one shared recurrent student trajectory with multi-depth endpoint supervision, so that exposed prefixes become answer-capable while remaining the exact continuation states for deeper computation. Decision distillation, view-local ordered alignment, and monotonic refinement provide secondary structure during hierarchy training. After the hierarchy is frozen, student-only capacity and confidence modules decide whether to accept the current endpoint, continue the cached trajectory, or fall back to explicit chain-of-thought reasoning; teacher and candidate-pool artifacts are absent at deployment.

threshold selection from becoming another fitting stage. Accordingly, terminal-only versus Core NEST isolates endpoint training, whereas routing and calibration comparisons concern allocation.

3.2 Core NEST: one state, two roles

After encoding x, the student computes one recurrent trajectory

$$H_x \to \mathbf{h}_1^{\mathrm{S}} \to \mathbf{h}_2^{\mathrm{S}} \to \cdots \to \mathbf{h}_{K_{\max}}^{\mathrm{S}}, \qquad \mathcal{K} = \{1, 2, 4, 8, 16\}.$$

The state $\mathbf{h}_k^{\mathrm{S}}$ is computed once and reused by every endpoint whose depth is at least k. Increasing the budget from K to $K' > K$ therefore extends the cached trajectory rather than constructing a depth-specific branch.

At an exposed depth K, the ordinary answer decoder must generate the complete target answer from the shared prefix $\mathbf{h}_{1:K}^{\mathrm{S}}$. Let $\mathcal{L}_{\mathrm{ans}}^{(K)}$ denote that token-level answer CE. A subset of depths $\mathcal{S} \subseteq \mathcal{K}$ is sampled from the fixed depth-view distribution P_{depth}, with within-view weights $\widetilde{w}_{K|\mathcal{S}}$. Core NEST is defined directly by the stochastic training objective

$$\mathcal{L}_{\mathrm{nested}} = \mathbb{E}_{\mathcal{S} \sim P_{\mathrm{depth}}} \left[\sum_{K \in \mathcal{S}} \widetilde{w}_{K|\mathcal{S}} \, \mathcal{L}_{\mathrm{ans}}^{(K)} \right]. \tag{1}$$

One state, two roles. A state $\mathbf{h}_k^{\mathrm{S}}$ has a single identity: it is part of the answer endpoint at the currently exposed depth, and it is also the exact continuation state from which every deeper endpoint is generated. Shallow endpoint CE therefore makes the shared prefix answer-capable *now*; deeper endpoint losses simultaneously require the same states to remain useful for *further computation*. This coupling is the core of NEST. It differs from standard deep supervision and early exiting, where an intermediate prediction is typically attached to a feed-forward layer through an auxiliary readout rather than being the recurrent continuation that generates all later endpoints. Core NEST is Eq. 1; the remaining losses below are structure-enhancing components rather than separate conceptual contributions.

Why the shared recurrence matters. Depth losses share both parameters and activations: a loss at $K = 2$ updates states later reused at $K = 4, 8, 16$, while deeper losses backpropagate through the same early states. NEST is therefore one computation whose prefixes become progressively usable, not a collection of separate solvers. Fixed-prefix evaluation directly measures this intended property independently of the learned routing policy.

3.3 STRUCTURE-ENHANCING SUPERVISION

The auxiliary terms are designed to make the shared hierarchy smoother and easier to route, not to create independent trajectories. For open-ended outputs, teacher and student compare decisions on one frozen candidate set $\mathcal{C}(x)$ built offline from the question only. Candidate answers are scored by length-normalized sequence log-likelihood, $s(c) = L_c^{-1} \sum_t \log p(c_t \mid \cdot)$, so shorter answers are not automatically favored. Teacher and student scores are normalized over the same $\mathcal{C}(x)$ using a fixed distillation temperature T_{KD}; their candidate-set conditional KL provides decision distillation. Appendix A.3 gives the exact construction and normalization.

Why the auxiliary signals are secondary. Endpoint CE defines the core anytime objective. The auxiliary terms address narrower failure modes: decision distillation preserves teacher preferences among plausible answers, ordered alignment supplies a coarse progress signal, and monotonic refinement discourages deeper endpoints from moving farther from the teacher decision. None creates a depth-specific student branch; they only regularize how competence is organized along the shared trajectory. This is why experiments report Core NEST separately from the full system.

For the full model, the Stage-B objective on an exposed view $\mathcal{S} = \{K_1 < \cdots < K_m\}$ is

$$\mathcal{L}_{\mathrm{B}} = \mathbb{E}_{\mathcal{S}\sim P_{\mathrm{depth}}}\left[\sum_{K\in\mathcal{S}} \widetilde{w}_{K|\mathcal{S}}\left(\lambda_{\mathrm{ans}}\mathcal{L}_{\mathrm{ans}}^{(K)} + \lambda_{\mathrm{KD}}\mathcal{L}_{\mathrm{KD}}^{(K)} + \lambda_{\mathrm{step}}\mathcal{L}_{\mathrm{step}}^{(K)}\right) + \lambda_{\mathrm{mono}}\mathcal{L}_{\mathrm{mono}}(\mathcal{S})\right]. \quad (2)$$

Here $\mathcal{L}_{\mathrm{KD}}^{(K)} = T_{\mathrm{KD}}^2 D_{\mathrm{KL}}(q^{\mathrm{T}} \| q_K^{\mathrm{S}})$; with $a_K(k) = \lceil kR/K \rceil$, the ordered step loss is $K^{-1}\sum_{k=1}^{K} \|P\mathbf{h}_k^{\mathrm{S}} - \mathrm{sg}(\mathbf{h}_{a_K(k)}^{\mathrm{T}})\|_2^2$; and $\mathcal{L}_{\mathrm{mono}}$ penalizes positive increases in $D_K = \|\mathbf{z}_K^{\mathrm{S}} - \mathbf{z}^{\mathrm{T}}\|_\infty$ across adjacent exposed depths. Appendix A.5 gives the exact normalization and edge cases.

3.4 SUFFICIENT DEPTH AND CALIBRATED ADAPTIVE EXECUTION

After the hierarchy is frozen, Stage C learns how much computation an input needs. On router-fitting records, let $\mathbf{z}^{\mathrm{T}}$ and $\mathbf{z}_K^{\mathrm{S}}$ be teacher and student candidate log-probability vectors on the same frozen $\mathcal{C}(x)$, let m^{T} be the teacher top-1/top-2 candidate margin, and let $D_K = \|\mathbf{z}_K^{\mathrm{S}} - \mathbf{z}^{\mathrm{T}}\|_\infty$. With a fixed positive slack ξ, the supervision target is

$$K^\star = \min\{K \in \mathcal{K} : 2D_K \le m^{\mathrm{T}} - \xi\}. \quad (3)$$

This criterion preserves the teacher top-1 only inside the frozen finite candidate set; it is a router-training target, not a correctness guarantee for unrestricted generation.

Deployment is fully pool-free. A capacity predictor proposes an initial depth from student-side question features. At each reached endpoint, a confidence head reads only student states, first-answer-token margin/entropy, neighboring-depth distribution change, and latent-state dynamics. It either accepts the endpoint, continues the cached trajectory to the next depth, or falls back to explicit CoT. The full answer is decoded only after acceptance. Exact feature definitions and calibration pseudocode are deferred to Appendix B, where training-only and deployed quantities are kept separate.

Competence is not confidence. A strong shallow endpoint is useful only when the policy can recognize that it is sufficient, while calibration cannot make an intrinsically weak endpoint correct. NEST therefore separates competence from allocation. Teacher-side candidates and $K^\star$ supervise execution-module fitting but are never queried online; deployed accept/continue decisions use only student-side quantities. This prevents early stopping from requiring teacher calls, beam search, or correctness checks that could erase the saved compute.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP.

We evaluate on eight reasoning benchmarks—GSM8K, MATH-500, SVAMP, BBH, MultiArith, AQuA-RAT, PrOntoQA, and ProsQA—with Qwen3.5-4B as the primary backbone; DeepSeek-R1-Distill-Qwen-7B and DeepSeek-R1-Distill-Llama-8B are used for cross-backbone transfer. We report

Table 1: Five-run results on all eight Qwen3.5-4B reasoning benchmarks. Task-level accuracy cells are mean±sample std across runs.

Method	GSM8K	MATH-500	SVAMP	BBH	MultiArith	AQuA-RAT	PrOntoQA	ProsQA	Avg.K	Rel.GPU↓	Fallback (%)	Unsafe (%)↓
Direct Answer	86.3±0.2	62.4±0.2	82.8±0.5	67.6±0.4	88.5±0.2	64.7±0.4	77.1±0.4	72.2±0.7	–	0.245	–	–
Explicit CoT	92.5±0.1	68.1±0.3	88.6±0.3	72.2±0.3	93.5±0.4	70.8±0.2	84.3±0.4	79.6±0.3	–	1.004	–	–
Coconut	89.8±0.2	64.8±0.6	85.7±0.3	68.9±0.4	90.9±0.4	66.1±0.5	80.5±0.5	75.3±0.4	–	–	–	–
CODI	91.7±0.2	66.3±0.3	87.4±0.2	70.0±0.1	91.9±0.3	68.8±0.6	82.2±0.4	77.6±0.4	6.27	0.483	–	–
SIM-CoT	92.8±0.2	67.6±0.4	89.0±0.3	71.5±0.3	93.0±0.1	70.3±0.6	84.2±0.5	78.9±0.4	8.07	0.451	–	–
SwiReasoning	**93.5±0.2**	69.0±0.2	89.4±0.2	73.1±0.5	93.9±0.3	71.9±0.5	85.3±0.6	80.7±0.5	6.44	0.592	13.02	13.24
Fixed-$K = 8$	92.8±0.3	68.5±0.6	89.1±0.2	72.0±0.4	93.4±0.3	71.1±0.5	84.6±0.4	79.7±0.3	8.00	0.462	–	–
NEST	93.2±0.4	**69.7±0.6**	**90.2±0.2**	**74.2±0.4**	**94.7±0.3**	**72.6±0.4**	**85.8±0.4**	**81.0±0.3**	**5.30**	**0.402**	14.35	**12.30**

final-answer accuracy together with Avg.K, measured Rel.GPU, fallback, and unsafe latent error, and use held-out calibration for all adaptive acceptance thresholds. Appendix E describes the datasets, baseline families, data splits, evaluation metrics, hardware/protocol details, and statistical testing, while Appendix E.3 records the complete training and candidate-generation hyperparameters used by the reported runs.

4.2 Experimental evidence and research questions

We keep the main paper focused on the evidence needed to evaluate the central claim. RQ1 establishes the end-to-end frontier; RQ2 tests the endpoint-supervision mechanism under matched execution; RQ3 tests transfer beyond the native implementation; RQ4 checks deployability. RQ1 asks whether the final system is useful; RQ2 asks whether the core training intervention actually changes the competence profile of the shared trajectory; RQ3 asks whether that intervention survives when inserted into other latent-reasoning frameworks and model families; RQ4 checks whether the trained hierarchy remains useful after removing teacher and candidate artifacts from deployment. Together, these experiments distinguish system-level gains from mechanism evidence, transfer evidence, and deployment validity. Secondary sensitivity, calibration diagnostics, and scaling studies are organized by purpose in Appendix D.

RQ1: Does NEST improve the end-to-end accuracy–computation frontier, including under matched data/teacher/compute controls?

RQ2: Does multi-depth endpoint supervision improve shallow-prefix competence when the halting machinery is held fixed?

RQ3: Does the effect transfer across latent-reasoning frameworks and model families?

RQ4: Can pool-free, strictly calibrated deployment reliably exploit the learned hierarchy?

Evaluation protocol. All main comparisons use the same eight reasoning benchmarks. Qwen3.5-4B comparisons and causal ablations use five runs; cross-backbone and scaling studies use three. We report accuracy, Avg.K, and measured relative GPU-seconds/query, plus fallback and unsafe latent error for adaptive systems. Thresholds are selected only on the held-out calibration split and frozen before test evaluation. Full statistical, hardware, and data-role details are in Appendix E.

4.3 RQ1: End-to-end accuracy–computation frontier

Table 1 reports the eight-benchmark system comparison. NEST reaches 82.68 average accuracy with Avg.$K = 5.30$, Rel.GPU 0.402, Fallback 14.35%, and Unsafe 12.30%; SwiReasoning reaches 82.07 with 6.44, 0.592, 13.02%, and 13.24%. The operating point therefore gains 0.61 accuracy points while reducing relative GPU cost by 32.1%. Because training-side supervision differs across method families, Table 1 is an end-to-end system comparison rather than a causal supervision ablation.

A matched data/teacher/compute comparison controls the strongest adaptive baseline more tightly: both systems use the same core examples, backbone, teacher checkpoint, question-only trace pool, answer supervision, optimizer-step budget, and measured training-compute budget. Auxiliary targets remain method-specific, so this is still not a fully information-matched causal ablation. Under this control, SwiReasoning reaches 82.35 ± 0.18 average accuracy with Rel.GPU 0.565, while NEST reaches 82.68 ± 0.16 with Rel.GPU 0.402: $+0.33$ accuracy points at 28.8% lower GPU cost.

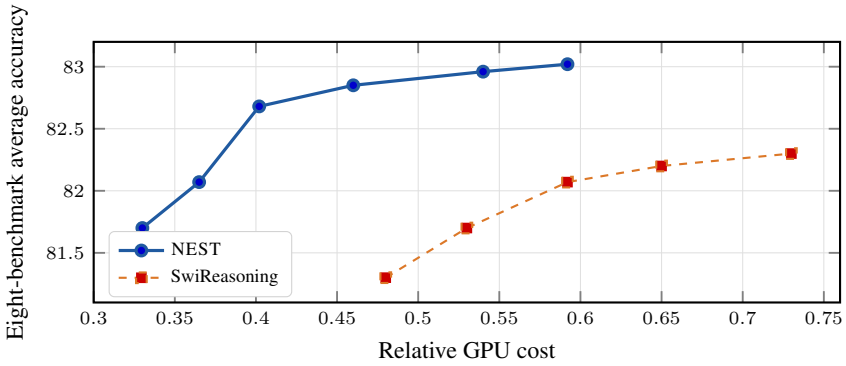


Figure 3: Accuracy–compute frontier. Acceptance thresholds are selected only from held-out calibration data and frozen before test evaluation. The matched-accuracy and matched-cost values reported in the text are post-hoc descriptive readouts from the already evaluated test frontier, using piecewise-linear interpolation between neighboring frontier points when required; they are not deployment thresholds chosen using test labels.

Statistical summary. Using the unrounded eight benchmark means underlying Table 1, the paired benchmark-level difference between NEST and SwiReasoning is $+0.61$ accuracy points (95% CI $[0.27, 0.94]$; paired $t(7) = 4.30$, $p = 0.0036$). This benchmark-level calculation is a descriptive cross-benchmark consistency check; the controlled five-run comparisons use the separate run-level summary-statistic checks specified in Appendix E. For the matched data/teacher/compute control, a Welch summary-statistic check computed from the five-run aggregate mean and sample standard deviation gives a $+0.33$-point difference (95% CI $[0.08, 0.58]$, $p = 0.0157$).

Figure 3 shows that the claim is not tied to one selected threshold. Thresholds are swept on held-out calibration data and frozen before test evaluation. Only after the complete frozen-threshold test frontier has been evaluated do we form matched-accuracy and matched-cost comparisons. These are post-hoc descriptive frontier readouts (with piecewise-linear interpolation between neighboring evaluated frontier points when needed), and they are never used to select a threshold, checkpoint, or model. On this basis, the SwiReasoning-level accuracy of 82.07 corresponds to Rel.GPU 0.365 for NEST versus 0.592 for SwiReasoning (38.3% lower); at the SwiReasoning cost reference Rel.GPU 0.592, the NEST frontier reads 83.02 average accuracy ($+0.95$ points). Thus the numerical matches to 82.07 and 0.592 are comparison coordinates defined after test evaluation, not calibration targets.

Answer to RQ1. NEST shifts the full accuracy–compute frontier: it is more accurate and cheaper than SwiReasoning at the primary operating point. Post-hoc readout of the frozen-threshold frontier gives 38.3% less GPU compute at matched accuracy and a 0.95-point gain at matched GPU cost.

4.4 RQ2: Mechanism—shallow-prefix competence under matched execution

Figure 4 directly measures endpoint competence before routing. At $K = 1$, terminal-only training, Core NEST, and full NEST reach 67.63, 73.58, and 76.43 average accuracy; at $K = 16$, they nearly converge to 82.43, 82.51, and 82.68. The gap therefore shrinks as computation increases. The shrinking gap shows that terminal-only training leaves much weaker shallow prefixes in the matched setting and that multi-depth endpoint CE moves usable competence earlier along the same trajectory.

Table 2 removes the strongest alternative explanation by holding the capacity predictor, confidence head, router-fitting data, and calibration fixed. Terminal-only, Core NEST, and full NEST reach 79.90 ± 0.21, 80.70 ± 0.19, and 82.68 ± 0.16 average accuracy, while Avg.K falls from 8.90 to 6.55 to 5.30. Thus better halting machinery cannot substitute for training the states on which the system halts.

Using the reported five-run aggregate mean and sample standard deviation as a run-level summary-statistic check, Core NEST improves over terminal-only training by $+0.80$ points (95% CI

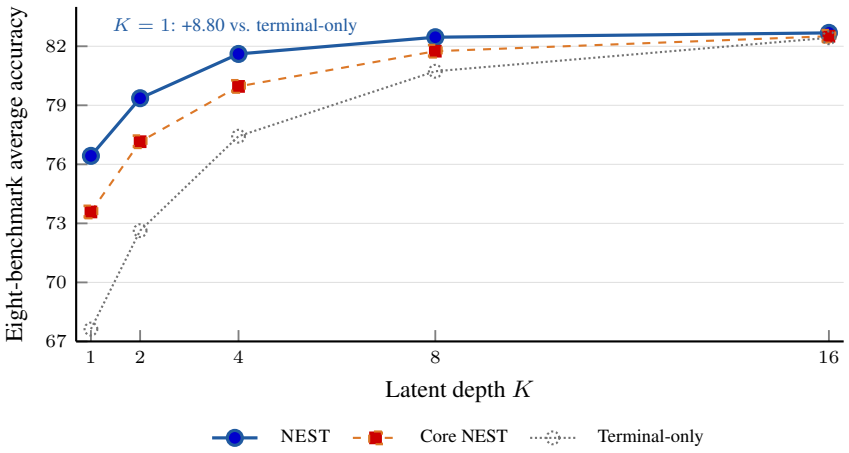


Figure 4: Fixed-prefix average accuracy over all eight benchmarks. Multi-depth endpoint supervision primarily improves shallow prefixes, while terminal-depth performance changes little.

Table 2: Five-run matched-router causal study over all eight benchmarks. Router architecture, router-fitting data, and calibration are identical; only hierarchy training changes.

Hierarchy	Avg. Acc.↑	Avg.K ↓	Rel.GPU↓	Unsafe↓
Terminal-only	79.90 ± 0.21	8.90	0.520	15.90%
Core NEST	80.70 ± 0.19	6.55	0.455	14.60%
Full NEST	82.68 ± 0.16	**5.30**	**0.402**	**12.30%**

$[0.51, 1.09]$, Welch $p = 2.4 \times 10^{-4}$), and full NEST improves over Core NEST by a further $+1.98$ points (95% CI $[1.72, 2.24]$, Welch $p = 1.4 \times 10^{-7}$).

The appendix retains the full component matrix, fixed-$K_{\max}$, random matched-cost, oracle, and ordered-alignment controls. Together they separate endpoint competence from allocation quality and show that the auxiliary terms refine, rather than replace, the multi-depth endpoint mechanism.

Answer to RQ2. Multi-depth endpoint supervision moves competence toward shallow prefixes; the effect remains when execution capacity and calibration are held fixed, while terminal-depth accuracy changes only marginally.

4.5 RQ3: Transfer across latent-reasoning frameworks

We insert Core NEST into SIM-CoT and CODI while keeping each host architecture and its execution-policy architecture, router-fitting split, acceptance rule, and calibration protocol fixed. Table 3 therefore isolates whether multi-depth endpoint supervision transfers beyond the native NEST implementation.

For SIM-CoT, average accuracy rises by 0.88 points while Avg.K falls by 21.3% and Rel.GPU by 9.1%. CODI shows the same direction: $+0.87$ points, 17.1% fewer latent steps, and 11.0% lower relative GPU cost. Because execution machinery is matched within each pair, the result is consistent with endpoint supervision making earlier recurrent prefixes more useful rather than merely substituting a stronger router. Appendix D further shows the same quality–compute direction on R1-Distill-Qwen-7B and R1-Distill-Llama-8B, with 30.2% and 26.9% lower Rel.GPU than SwiReasoning, respectively. These results show that the endpoint-supervision intervention transfers across the tested frameworks and backbones.

The corresponding five-run Welch summary-statistic checks give $+0.88$ points for SIM-CoT + Core NEST (95% CI $[0.63, 1.13]$, $p = 4.1 \times 10^{-5}$) and $+0.87$ points for CODI + Core NEST (95% CI $[0.59, 1.15]$, $p = 9.5 \times 10^{-5}$).

Table 3: Five-run controlled cross-framework test over all eight benchmarks. Within each host framework, baseline and +Core NEST use identical execution machinery; only multi-depth endpoint supervision is added.

Method	Avg. Acc.↑	Avg.K ↓	Rel.GPU↓
SIM-CoT	80.92 ± 0.18	8.07	0.451
SIM-CoT + Core NEST	81.80 ± 0.16	**6.35**	**0.410**
CODI	79.48 ± 0.20	6.27	0.483
CODI + Core NEST	80.35 ± 0.18	**5.20**	**0.430**

Answer to RQ3. Core NEST improves both SIM-CoT and CODI while reducing computation under matched execution machinery, and the same quality–compute direction persists on two additional backbones.

4.6 RQ4: Pool-free and strictly calibrated deployment

Deployment removes every training-only artifact: no candidate pool, teacher score/state, gold label, or oracle depth is available to the router. The pool-free implementation reaches 82.58 average accuracy, Rel.GPU 0.410, and 12.45% unsafe latent error, close to the primary 82.68/0.402/12.30% operating point. With hierarchy training, router/confidence fitting, threshold calibration, and testing strictly disjoint, NEST reaches 82.60 average accuracy, Avg.$K = 5.40$, Rel.GPU 0.410, Unsafe 12.40%, and ECE 0.020.

Held-out threshold calibration is not cosmetic: the uncalibrated policy raises coverage from 85.65% to 89.5% but increases unsafe latent error from 12.30% to 13.27%. Because calibration selects an operating threshold without updating the confidence head, score-level ECE, Brier, AUROC, and full-curve AURC are unchanged by this threshold selection. Detailed predictor error, risk–coverage, task-shift recalibration, and exact inference pseudocode are grouped in Appendix B and Appendix D.

Answer to RQ4. Pool-free inference preserves the operating regime, while disjoint held-out calibration materially reduces unsafe acceptance; under distribution shift, the threshold may require modest target-domain recalibration even when the hierarchy remains usable.

5 Conclusion and Future Work

The central empirical finding is that multi-depth endpoint supervision is an effective way to make recurrent latent trajectories more usable under adaptive computation. Explicitly training intermediate recurrent prefixes as answer endpoints before fitting execution policies improves shallow-prefix competence and the resulting accuracy–compute trade-off. Core NEST implements this design with nested endpoint supervision on a shared recurrent continuation, and the full system adds decision and trajectory structure plus calibrated execution. Fixed-prefix and matched-router controls separate the effect of endpoint training from halting-policy capacity, the matched data/teacher/compute comparison preserves the quality–compute advantage against the strongest adaptive baseline, and controlled cross-framework interventions show similar gains when Core NEST is inserted into SIM-CoT and CODI. Pool-free deployment and disjoint held-out calibration preserve the same operating regime without accessing training-time candidate artifacts. Across eight benchmarks, the resulting system shifts the accuracy–compute frontier; a post-hoc matched-accuracy readout of the frozen-threshold test frontier gives 38.3% less relative GPU cost at SwiReasoning-level accuracy. Future work should test substantially larger models, non-mathematical settings, and alternative latent-reasoning architectures.

Limitations. The present evidence is intentionally concentrated on reasoning benchmarks and a finite family of recurrent latent architectures, but they do not establish that multi-depth endpoint supervision is universally optimal. The finite candidate set used for training-side decision supervision is also an approximation to unrestricted generation, and the $K^\star$ criterion preserves teacher preference only inside that set. Finally, calibration can shift across domains even when the hierarchy remains useful, so deployment in substantially different distributions should reserve target-domain calibration data rather than reuse a source threshold uncritically.

AI USE STATEMENT

Generative AI tools were used for translation assistance, readability editing, and limited suggestions on manuscript organization and LaTeX formatting. They were not used to generate experimental data or execute the reported experiments. The authors reviewed all AI-assisted text and take responsibility for the final manuscript.

REFERENCES

Andrea Banino, Jan Balaguer, and Charles Blundell. PonderNet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021. URL https://arxiv.org/abs/2107.05407.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. URL https://arxiv.org/abs/2501.12948.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1807.03819.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/76917808731dae9e6d62c2a7a6afb542-Abstract-Conference.html.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016. URL https://arxiv.org/abs/1603.08983.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024. URL https://arxiv.org/abs/2412.06769.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems: Datasets and Benchmarks Track*, 2021. URL https://arxiv.org/abs/2103.03874.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. URL https://arxiv.org/abs/1503.02531.

Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pp. 562–570. PMLR, 2015. URL https://proceedings.mlr.press/v38/lee15a.html.

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*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/aca97732e30bcf1303bc22ac3924fd16-Abstract-Conference.html.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 158–167. Association for Computational Linguistics, 2017. doi: 10.18653/v1/P17-1015. URL https://aclanthology.org/P17-1015/.

Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. FastBERT: a self-distilling BERT with adaptive inference time. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 6035–6044. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.537. URL https://aclanthology.org/2020.acl-main.537/.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? In *Proceedings of NAACL-HLT*, pp. 2080–2094. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.naacl-main.168. URL https://aclanthology.org/2021.naacl-main.168/.

Qwen Team. Qwen3.5: Towards native multimodal agents. Official Qwen release and model card, February 2026. URL https://huggingface.co/Qwen/Qwen3.5-4B.

Subhro Roy and Dan Roth. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1743–1752. Association for Computational Linguistics, 2015. doi: 10.18653/v1/D15-1202. URL https://aclanthology.org/D15-1202/.

Abulhair Saparov and He He. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.01240.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/6fac9e316a4ae75ea244ddcef1982c71-Paper-Conference.pdf.

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. CODI: Compressing chain-of-thought into continuous space via self-distillation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 677–693. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.36. URL https://aclanthology.org/2025.emnlp-main.36/.

Dachuan Shi, Abedelkadir Asi, Keying Li, Xiangchi Yuan, Leyan Pan, Wenke Lee, and Wen Xiao. Swireasoning: Switch-thinking in latent and explicit for pareto-superior reasoning llms. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/ddb7bad46132a323aa9d039000941881-Abstract-Conference.html.

Mirac Suzgun, Nathan Scales, Nathanael Scharli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. 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. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-acl.824. URL https://aclanthology.org/2023.findings-acl.824/.

Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. BranchyNet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, 2016. doi: 10.1109/ICPR.2016.7900006. URL https://arxiv.org/abs/1709.01686.

Shubham Toshniwal, Ivan Moshkov, Sean Narenthiran, Daria Gitman, Fei Jia, and Igor Gitman. Openmathinstruct-1: A 1.8 million math instruction tuning dataset. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://arxiv.org/abs/2402.10176.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022. URL https://arxiv.org/abs/2211.14275.

Xiaoqiang Wang, Suyuchen Wang, Yun Zhu, and Bang Liu. System-1.5 reasoning: Traversal in language and latent spaces with dynamic shortcuts. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-2191. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/5e66b9f710a8e38351da690f3806cba9-Abstract-Conference.html.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. 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. URL https://arxiv.org/abs/2203.11171.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://arxiv.org/abs/2201.11903.

Xilin Wei, Xiaoran Liu, Yuhang Zang, Xiaoyi Dong, Yuhang Cao, Jiaqi Wang, Xipeng Qiu, and Dahua Lin. SIM-CoT: Supervised implicit chain-of-thought. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/5d087955ee13fe9a7402eedec879b9c3-Abstract-Conference.html.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2246–2251. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.204. URL https://aclanthology.org/2020.acl-main.204/.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, 2022. doi: 10.52202/068431-1126. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

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

Denny Zhou, Nathanael Scharli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. Least-to-most prompting enables complex reasoning in large language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2205.10625.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. BERT loses patience: Fast and robust inference with early exit. In *Advances in Neural Information Processing Systems*, volume 33, pp. 18330–18341, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/d4dd111a4fd973394238aca5c05bebe3-Abstract.html.

A TRAINING DETAILS FOR THE NESTED REASONING HIERARCHY

This appendix records implementation details that are useful for reproduction but not needed to define the core method. The notation follows the main text: $\mathbf{h}_k^{\mathrm{S}}$ is a student recurrent state, ρ_s^{T} is teacher-step text, and $\mathbf{h}_s^{\mathrm{T}}$ is the corresponding detached teacher representation.

A.1 SINGLE TRAJECTORY AND EXPOSED TRAINING VIEWS

The student computes one trajectory through $K_{\max}$. Exposing endpoint K means attaching the answer and auxiliary losses to the already-computed prefix $\mathbf{h}_{1:K}^{\mathrm{S}}$; it does not create a separate branch. If a minibatch exposes several depths, all of them reuse the same cached states.

Depth sampling defines the training objective itself. A subset $\mathcal{S} \sim P_{\mathrm{depth}}$ is drawn, and Eq. 1 is evaluated on that sampled view. The exact probabilities of P_{depth} belong to the run configuration and are reported with the optimization hyperparameters in Appendix E.

A.2 TEACHER TRACE AND PROCESS TARGETS

The teacher receives the question only and generates a reasoning trace followed by its own final answer. The benchmark gold answer is not inserted into the teacher prompt, and traces are not filtered by teacher correctness. Before process supervision is constructed, the final-answer span is removed. For the primary Qwen3.5-4B experiments, the frozen explicit teacher and recurrent student are initialized from the same Qwen3.5-4B checkpoint. The teacher copy remains frozen throughout artifact generation and hierarchy training, while the student copy is updated by NEST; thus the primary setting is self-distillation from the same base checkpoint rather than distillation from a larger teacher.

A deterministic parser produces ordered textual steps $\rho^{\mathrm{T}} = (\rho_1^{\mathrm{T}}, \ldots, \rho_R^{\mathrm{T}})$. It first splits non-empty lines; a single-line trace is split on sentence-final punctuation outside LaTeX math spans and balanced delimiters, with decimal points and common abbreviations protected. For each step, the final-layer teacher states over its token span are mean-pooled to obtain $\mathbf{h}_s^{\mathrm{T}}$. These teacher representations are detached or cached before student backpropagation.

A.3 LEAKAGE-SAFE CANDIDATE CONSTRUCTION

For every open-ended hierarchy-training or router-fitting record that uses candidate-based supervision, candidate generation runs offline on a label-stripped view containing only the example identifier and question. The frozen explicit teacher serves as the candidate generator. Generation uses 8-beam deterministic decoding plus 8 nucleus samples at each temperature in $\{0.4, 0.7, 1.0\}$, with `top_p` $= 0.95$, `max_new_tokens` $= 512$, and seed 1234.

A deterministic final-answer parser takes the final `\boxed{...}` span when present, then a GSM8K-style `####` answer, then an explicit `Final answer:`/`Answer:` field, and otherwise the last non-empty line. Extracted answers are normalized without consulting the gold label, deduplicated, and truncated to at most 32 unique candidates. When truncation is necessary, candidates are ranked only by the generator's length-normalized sequence score with exponent $\alpha = 1$.

The resulting $\mathcal{C}(x)$ is immutable. Teacher and every student depth score exactly the same set. With length-normalized scores s_K^{S} and s^{T} and a fixed distillation temperature $T_{\mathrm{KD}} > 0$, the candidate-set conditional distributions are

$$q_K^{\mathrm{S}}(c \mid x) = \frac{\exp(s_K^{\mathrm{S}}(c \mid x)/T_{\mathrm{KD}})}{\sum_{c' \in \mathcal{C}(x)} \exp(s_K^{\mathrm{S}}(c' \mid x)/T_{\mathrm{KD}})}, \quad q^{\mathrm{T}}(c \mid x) = \frac{\exp(s^{\mathrm{T}}(c \mid x)/T_{\mathrm{KD}})}{\sum_{c' \in \mathcal{C}(x)} \exp(s^{\mathrm{T}}(c' \mid x)/T_{\mathrm{KD}})}. \tag{4}$$

Decision distillation uses $T_{\mathrm{KD}}^2 D_{\mathrm{KL}}(q^{\mathrm{T}} \| q_K^{\mathrm{S}})$ on this shared finite coordinate system; it is not an exact KL over the infinite autoregressive sequence space. Gold answers are never appended, used for candidate filtering/ranking, or used to tune pool hyperparameters. If $|\mathcal{C}(x)| < 2$, ordinary endpoint CE remains active but candidate-based KD, monotonicity, margin, and $K^\star$ supervision are disabled. Candidate artifacts are absent from deployed inference.

A.4 VIEW-LOCAL ALIGNMENT ON SHARED STATES

The alignment map is indexed by the exposed endpoint: $a_K(k) = \lceil kR/K \rceil$. This notation is deliberate. The state $\mathbf{h}_k^{\mathrm{S}}$ is global and shared, whereas the teacher target selected for it is local to the current endpoint view K. When several depths are exposed in the same minibatch, a shared state can therefore receive several alignment gradients, each corresponding to a different normalized-progress view. No student state is duplicated or recomputed.

The stop-gradient operator used by the view-local alignment loss has identity forward value and zero derivative with respect to the teacher target, i.e., $\mathrm{sg}(\mathbf{h}_s^{\mathrm{T}}) = \mathbf{h}_s^{\mathrm{T}}$ in the forward pass and $\partial\,\mathrm{sg}(\mathbf{h}_s^{\mathrm{T}})/\partial\mathbf{h}_s^{\mathrm{T}} = 0$ in the backward pass. The learned projection P handles teacher/student hidden-dimension mismatch.

A.5 EXACT STAGE-B OBJECTIVE

Let an exposed training view be $\mathcal{S} = \{K_1 < \cdots < K_m\}$. The candidate-set decision-distillation term at depth K is

$$\mathcal{L}_{\mathrm{KD}}^{(K)} = T_{\mathrm{KD}}^2 D_{\mathrm{KL}}\big(q^{\mathrm{T}}(\cdot \mid x) \,\|\, q_K^{\mathrm{S}}(\cdot \mid x)\big)\,, \tag{5}$$

using the common frozen candidate set and the normalized distributions in Eq. 4. The view-local ordered alignment term is

$$\mathcal{L}_{\mathrm{step}}^{(K)} = \frac{1}{K}\sum_{k=1}^{K}\Big\|P\mathbf{h}_k^{\mathrm{S}} - \mathrm{sg}\Big(\mathbf{h}_{a_K(k)}^{\mathrm{T}}\Big)\Big\|_2^2, \qquad a_K(k) = \left\lceil \frac{kR}{K} \right\rceil. \tag{6}$$

The same student state may therefore receive alignment gradients from several exposed endpoint views, but the state itself is never duplicated. For candidate-log-probability vectors $\mathbf{z}_K^{\mathrm{S}}$ and $\mathbf{z}^{\mathrm{T}}$, define $D_K = \|\mathbf{z}_K^{\mathrm{S}} - \mathbf{z}^{\mathrm{T}}\|_\infty$. Monotonic refinement is

$$\mathcal{L}_{\mathrm{mono}}(\mathcal{S}) = \frac{1}{\max(1, m-1)}\sum_{j=1}^{m-1}\big[D_{K_{j+1}} - D_{K_j}\big]_+, \qquad [u]_+ = \max(0, u). \tag{7}$$

The complete hierarchy-training loss is exactly Eq. 2. If $|\mathcal{C}(x)| < 2$, the candidate-based KD and monotonic terms are disabled for that record while endpoint CE and available step alignment remain active. The weights used by the main runs are listed in Table 22.

A.6 STAGE-SPECIFIC OPTIMIZATION

Training is intentionally staged; there is no global "all losses" objective.

Stage A: teacher preprocessing. Generate teacher traces and, where needed, cache the label-independent candidate sets and teacher-side candidate scores/process targets.

Stage B: hierarchy training. Optimize only the Stage-B hierarchy objective described in Section 3 on hierarchy-training records. This stage updates the student hierarchy and the projection used by step alignment. It does not fit acceptance thresholds, the capacity predictor, or the confidence head.

Stage C: execution-module fitting. Freeze/checkpoint the hierarchy. On a dedicated router-fitting split, compute finite-candidate $K^\star$ labels using Eq. 3, then fit the capacity predictor and confidence head. A disjoint calibration split is used only after those modules are frozen to select the acceptance threshold. Final test evaluation occurs after threshold selection.

Algorithm A1 summarizes the complete staged training procedure and makes the separation among hierarchy training, execution-module fitting, and threshold calibration explicit.

Algorithm A1 Staged NEST training

Require: hierarchy-training data; frozen teacher; student parameters θ; depth-view distribution P_{depth}
Ensure: trained hierarchy, execution modules, and calibrated threshold
1: Offline, generate teacher traces and leakage-safe candidate artifacts
2: **for** each hierarchy-training minibatch **do**
3: Compute one student trajectory to the largest sampled depth
4: Sample exposed endpoint set $\mathcal{S} \sim P_{\text{depth}}$
5: **for** each $K \in \mathcal{S}$ **do**
6: Reuse prefix $\mathbf{h}^{\text{S}}_{1:K}$; compute endpoint CE, KD, and view-local alignment
7: **end for**
8: Compute monotonic refinement over the sorted exposed depths and update the Stage-B hierarchy parameters
9: **end for**
10: Freeze/checkpoint the hierarchy
11: Fit capacity/confidence modules only on router-fitting records
12: Freeze execution heads and select the acceptance threshold only on the disjoint calibration split

A.7 Operational definition of latent depth

One latent reasoning step is one additional autoregressive latent position whose input embedding is the previous latent terminal hidden state. Continuous states bypass vocabulary projection and token re-embedding between latent positions. Hence K counts recurrent hidden-state feedback updates, not Transformer layers, text tokens, or hidden dimensions. With KV caching, extending from $K = 4$ to $K = 8$ reuses the question prefix and $\mathbf{h}^{\text{S}}_{1:4}$ and evaluates only the four new latent positions.

B Adaptive Execution After Hierarchy Training

Adaptive execution is fitted only after the Stage-B hierarchy is frozen. This appendix specifies the student-only features used at deployment and the held-out calibration rule.

B.1 Capacity predictor

Before recurrent latent computation, the predictor uses a mean-pooled student question representation $\mathbf{h}^{\text{S}}_{x,\text{pool}}$ together with zero-latent first-token uncertainty. For any reached depth K, let $\pi^{\text{S}}_K = \text{softmax}(\ell^{\text{S}}_K)$. We define

$$m^{\text{S}}_K = \log(\pi^{\text{S}}_K)^{(1)} - \log(\pi^{\text{S}}_K)^{(2)}, \qquad H^{\text{S}}_K = -\frac{1}{\log|V|}\sum_{v\in V}\pi^{\text{S}}_K(v)\log\pi^{\text{S}}_K(v), \tag{8}$$

where (1) and (2) denote the two largest first-token probabilities. The predictor uses the zero-latent values m^{S}_0 and H^{S}_0. The input feature is

$$\phi(x) = [\mathbf{h}^{\text{S}}_{x,\text{pool}}; m^{\text{S}}_0; H^{\text{S}}_0],$$

and a small MLP predicts a class in $\mathcal{K} \cup \{F\}$, where F means immediate explicit fallback. For router-fitting records, the target class is $K^\star$ when it exists and F otherwise. This module is trained only on the router-fitting split.

B.2 Confidence after latent computation

At reached depth K, the endpoint produces π^{S}_K before any complete answer is decoded. In addition to m^{S}_K and H^{S}_K from Eq. 8, the confidence head uses

$$J^{\text{S}}_K = D_{\text{JS}}(\pi^{\text{S}}_K \,\|\, \pi^{\text{S}}_{K^-}), \qquad \Delta_K = \|\mathbf{h}^{\text{S}}_K - \mathbf{h}^{\text{S}}_{K-1}\|_2, \qquad \delta_K = 1 - \cos(\mathbf{h}^{\text{S}}_K, \mathbf{h}^{\text{S}}_{K-1}). \tag{9}$$

A representative confidence feature vector is

$$[\mathbf{h}^{\text{S}}_K; m^{\text{S}}_K; H^{\text{S}}_K; J^{\text{S}}_K; \Delta_K; \delta_K].$$

During router fitting only, the endpoint is fully decoded to obtain a correctness label for binary confidence-head training. After fitting, the confidence head is frozen. At deployment, rejected endpoints are not fully decoded; the cached trajectory is simply extended.

B.3 CALIBRATION AND SELECTIVE RISK

For acceptance threshold τ_{acc}, let $A_{\tau_{\mathrm{acc}}}(x)$ indicate latent acceptance. We calibrate the system-level unsafe event

$$\ell_{\tau_{\mathrm{acc}}}(x, y) = \mathbf{1}[A_{\tau_{\mathrm{acc}}}(x) = 1 \ \wedge \ \hat{y}_{\mathrm{latent}} \neq y]. \tag{10}$$

The corresponding risk is $R(\tau_{\mathrm{acc}}) = \mathbb{E}[\ell_{\tau_{\mathrm{acc}}}]$. On a finite threshold grid $\mathcal{T}$ with M values and N independent calibration records, a simple uniform Hoeffding bound is

$$R(\tau) \leq \widehat{R}_N(\tau) + \sqrt{\frac{\log(M/\delta)}{2N}} \quad \text{for all } \tau \in \mathcal{T} \tag{11}$$

with probability at least $1 - \delta$. Threshold selection maximizes latent coverage among feasible thresholds. Importantly, the calibration records never update the confidence-head parameters.

B.4 INFERENCE ALGORITHM

Algorithm A2 gives the complete pool-free inference procedure. It makes the cache reuse, accept/continue decision, and explicit fallback path explicit while keeping all teacher-side artifacts outside deployed inference.

Algorithm A2 Pool-free adaptive execution with cached latent continuation

Require: Input x; frozen student hierarchy; capacity predictor; confidence head; calibrated threshold $\tau^\star$; ladder $\mathcal{K}$
Ensure: Final answer $\hat{y}$
Encode x, compute π_0^{S}, and predict initial class $\hat{c}_0$
if $\hat{c}_0 = F$ **then**
    Run explicit CoT and **return** its final answer
end if
$K \leftarrow \hat{c}_0$; initialize the cached shared student trajectory
while true **do**
    Extend the cached trajectory only as needed to reach depth K
    Compute π_K^{S} and student-only confidence features
    Evaluate confidence c_K
    if $c_K \geq \tau^\star$ **then**
        Greedily decode the complete endpoint answer $\hat{y}_K$ and **return** $\hat{y}_K$
    else if $K < K_{\max}$ **then**
        Cache π_K^{S} and move to the next depth in $\mathcal{K}$
    else
        Run explicit CoT and **return** its final answer
    end if
end while

The inference inputs contain no teacher trace, teacher logits/states, gold answer, oracle depth, cached candidate pool, beam search, or self-consistency samples.

B.5 FINITE-GRID CALIBRATION ALGORITHM

Algorithm A3 instantiates the held-out threshold-selection rule used by the execution policy. The algorithm operates only on frozen confidence scores and calibration correctness labels; it does not update the hierarchy or confidence-head parameters.

Algorithm A3 Finite-grid calibration of latent acceptance

Require: Calibration scores $\{c_i\}_{i=1}^N$; correctness labels $\{b_i\}_{i=1}^N$; threshold grid $\mathcal{T}$; target unsafe rate ϵ; confidence level δ
Ensure: Calibrated threshold $\tau^\star$ or no feasible threshold
$M \leftarrow |\mathcal{T}|$ and $r_N \leftarrow \sqrt{\log(M/\delta)/(2N)}$
$\mathcal{F} \leftarrow \emptyset$
for each $\tau \in \mathcal{T}$ **do**
  Compute empirical unsafe risk $\widehat{R}_N(\tau)$ and latent coverage $\widehat{\mathrm{Cov}}(\tau)$
  if $\widehat{R}_N(\tau) + r_N \le \epsilon$ **then**
    Add τ to $\mathcal{F}$
  end if
end for
if $\mathcal{F} = \emptyset$ **then**
  return no feasible latent-acceptance threshold
else
  return $\tau^\star \leftarrow \arg\max_{\tau \in \mathcal{F}} \widehat{\mathrm{Cov}}(\tau)$
end if

The output of Algorithm A3 is the single threshold $\tau^\star$ consumed by Algorithm A2. Thus B.5 is the calibration stage that connects the frozen confidence model to the deployed accept/continue rule.

B.6 Computation accounting

Reported inference cost includes all latent computation already spent before any explicit fallback. The experiments report measured GPU-seconds and wall-clock latency under a shared harness; Avg.K is not treated as a substitute for either quantity. One-time model-training overhead is reported separately in Appendix D; offline candidate-pool preprocessing is disclosed separately and is not included in deployed inference cost.

C Proofs and Theoretical Details

The theory is intentionally narrow. It justifies the finite-candidate supervision target used to fit the execution modules; it does not claim correctness over unrestricted sequence generation.

C.1 Finite-candidate top-1 preservation

Let $y^{\mathrm{T}} = \arg\max_c z_c^{\mathrm{T}}$, let $m^{\mathrm{T}} = z_{y^{\mathrm{T}}}^{\mathrm{T}} - \max_{c \neq y^{\mathrm{T}}} z_c^{\mathrm{T}}$ be the teacher top-1/top-2 candidate margin, and let $D_K = \|\mathbf{z}_K^{\mathrm{S}} - \mathbf{z}^{\mathrm{T}}\|_\infty$. For any competing candidate $c \neq y^{\mathrm{T}}$,

$$z_{K,y^{\mathrm{T}}}^{\mathrm{S}} - z_{K,c}^{\mathrm{S}} \ge (z_{y^{\mathrm{T}}}^{\mathrm{T}} - z_c^{\mathrm{T}}) - 2D_K \ge m^{\mathrm{T}} - 2D_K.$$

Hence $2D_K < m^{\mathrm{T}}$ is sufficient for the student endpoint to preserve the teacher's top-1 candidate. The slack ξ in Eq. 3 simply makes the router-fitting target more conservative.

C.2 Why the earliest sufficient depth is the relevant target

When execution cost is nondecreasing with depth, the smallest depth satisfying the sufficient condition cannot cost more than any deeper satisfying depth. This elementary observation motivates using the minimum in Eq. 3; no stronger optimization theorem is required.

The monotonic refinement term serves a different role. It encourages teacher–student candidate distortion not to increase as more recurrent computation is appended, which makes the set of candidate sufficient depths less erratic. The $K^\star$ definition itself remains valid even when distortion is non-monotone.

C.3 FALLBACK-AWARE COMPUTATION

Efficiency claims include computation already spent before fallback. If F denotes explicit fallback with probability p_F, a conservative accounting is

$$\mathbb{E}[C_{\text{adaptive}}] = (1 - p_F)\mathbb{E}[C(K) \mid \neg F] + p_F[C(K_{\max}) + C_{\text{explicit}}] .$$

Measured GPU-seconds and wall-clock latency, rather than latent-step count alone, are reported in the experiments. One-time model-training overhead is reported separately in the supplementary efficiency analysis.

C.4 LIMITATIONS OF THE GUARANTEE

The preservation result transfers the teacher's top-1 decision only on the frozen finite candidate set. If the teacher is wrong, preservation does not make the student correct; and for open-ended answers the statement does not cover sequences absent from $\mathcal{C}(x)$. It is therefore a router-supervision device rather than an oracle available at test time. Deployed inference instead uses student-only confidence features and held-out calibration.

D SUPPLEMENTARY EXPERIMENTS AND DETAILED ANALYSIS

D.1 collects mechanism and causal controls, D.2 transfer/robustness, D.3 adaptive execution/calibration, and D.4 efficiency/stress tests.

D.1 CORE MECHANISM AND CAUSAL CONTROLS

D.1.1 RELATION TO DEEP SUPERVISION AND EARLY EXITS

Deep supervision and early exiting are direct conceptual antecedents, and we do not treat intermediate supervision itself as novel. Deeply-Supervised Nets place companion losses on intermediate feed-forward layers (Lee et al., 2015); BranchyNet and later language-model exits attach classifiers to intermediate network depth (Teerapittayanon et al., 2016; Xin et al., 2020; Liu et al., 2020; Zhou et al., 2020). In NEST, the nesting variable is recurrent latent reasoning depth. A prefix $\mathbf{h}^{\mathrm{S}}_{1:K}$ is consumed by the answer readout at depth K and remains the exact recurrent state from which every deeper endpoint is generated. The contribution is therefore the coupling of endpoint competence and shared recurrent continuation, followed by calibrated execution, rather than the generic idea of placing losses at multiple depths.

A clean multi-depth-answer-only control isolates the effect of endpoint supervision from the remaining structured terms. It retains $\sum_K w_K \mathcal{L}^{(K)}_{\text{ans}}$ and removes KD, step alignment, and monotonic refinement. Table 4 shows the cumulative quality–compute effect, and Table 5 localizes that effect across fixed prefix depths. The computational distinction is simple: every shallower endpoint is also the exact state prefix from which deeper recurrent computation continues.

Table 4: Five-run cumulative training controls averaged over all eight benchmarks under the same common-router evaluation protocol. Core NEST is the multi-depth endpoint-CE-only objective and is the same Core-NEST condition used in the matched-router study.

Training objective	Avg. Acc.↑	Avg.K ↓	Rel.GPU↓	Unsafe↓
Terminal-only	79.90	8.90	0.520	15.90%
Core NEST (multi-depth CE)	80.70	6.55	0.455	14.60%
+ decision KD	81.34	6.42	0.437	13.90%
+ ordered step alignment	82.33	5.75	0.420	12.85%
NEST full	82.68	5.30	0.402	12.30%

Tables 4 and 5 separate endpoint competence from the auxiliary structure. Direct endpoint supervision explains most of the transfer toward small K: at $K = 1$, it closes 5.95 of the 8.80 points between terminal-only training and the full model. Decision distillation, ordered semantic alignment, and monotonic refinement then improve how that competence is organized along the shared continuation trajectory. The same prefix must support a valid answer now and remain useful as the starting state for deeper computation.

Table 5: Fixed-prefix average accuracy over all eight benchmarks for the clean deep-supervision control.

Training formulation	$K = 1$	$K = 2$	$K = 4$	$K = 8$	$K = 16$
Terminal-only	67.63	72.63	77.43	80.73	82.43
Core NEST (multi-depth CE)	73.58	77.15	79.96	81.75	82.51
NEST full	76.43	79.36	81.61	82.46	82.68

D.1.2 Why ordered step alignment matters empirically

Table 6 isolates ordering in the step-alignment signal. Random teacher-state matching gives little improvement over removing the term. Unordered nearest matching provides an intermediate gain. Relative-position ordered alignment reaches the best eight-benchmark average accuracy and the strongest trajectory structure.

Table 6: Controlled step-alignment comparison averaged over all eight benchmarks.

Alignment	Avg. Acc.↑	Pairwise Cos.↓	Eff. Rank↑	Monotonicity↑
None	81.19	0.82	4.4	76%
Random teacher state	81.34	0.78	4.9	78%
Unordered nearest matching	81.77	0.70	6.3	84%
Relative-position ordered	82.68	0.56	8.8	92%

The ordered variant improves endpoint quality together with lower state similarity, higher effective rank, and stronger adjacent-depth refinement. The comparison separates two possible explanations for the step loss. If the benefit came only from adding another representation penalty, random matching would have recovered a similar gain. It does not. The improvement increases as the target correspondence becomes more consistent with reasoning order, which links the accuracy change to the temporal structure of the supervision signal without implying one-to-one semantic equivalence between latent states and human reasoning steps.

The representation statistics point in the same direction. Random and unordered targets leave the state sequence more redundant, while relative-position alignment expands effective rank and reduces pairwise cosine similarity. This suggests that ordered targets shape how information is distributed across recurrent steps before the change is visible at the answer endpoint. The result also explains why removing step supervision raises Avg.K: a less structured trajectory needs more recurrent updates before a reliable endpoint emerges.

Full ablation matrix Table 7 separates hierarchy-training components from execution-policy components. Removing step supervision causes the largest accuracy drop among the representation terms, while removing nested training produces the largest increase in Avg.K. Removing calibration or explicit fallback has a smaller effect on the endpoint model itself but sharply increases unsafe latent error, showing that hierarchy quality and execution reliability are distinct parts of the full system.

The fixed-$K_{\max}$, random matched-cost, and oracle rows provide three useful controls. Fixed maximum depth shows that the full hierarchy can retain terminal quality with much less computation; random matched-cost allocation shows that the gain is not explained by average compute alone; and the oracle row quantifies the remaining headroom in depth prediction.

Trajectory diagnostics The trajectory diagnostics measure how recurrent latent states evolve across computation depth. They quantify teacher–student distortion, representation diversity, effective rank, and adjacent-depth monotonicity. We use them as supporting diagnostics of an operational computation hierarchy; they do not establish that individual latent states correspond one-to-one with human-interpretable reasoning steps.

Tables 8 and 9 provide complementary trajectory diagnostics. The distortion table tracks how endpoint decisions approach the teacher reference with depth, while the structure table measures redundancy and dimensional diversity inside the recurrent state sequence. Together, the measurements show that lower state similarity, higher effective rank, and more monotone teacher alignment accompany stronger shallow-prefix competence.

Table 7: Five-run ablation study averaged equally over all eight benchmarks.

Variant	Avg. Acc.↑	Avg.K ↓	Rel.GPU↓	Unsafe↓
NEST	82.68	5.30	0.402	12.30%
NEST − self-distillation	81.52	6.18	0.425	13.95%
NEST − step supervision	81.19	6.79	0.441	14.55%
NEST − nested training	81.68	7.58	0.487	14.05%
NEST − monotonicity	82.39	5.87	0.415	12.90%
Fixed $K = K_{\max} = 16$	82.68	16.00	0.575	17.32%
Entropy-only routing	82.13	6.35	0.479	12.62%
Margin-only routing	82.03	6.21	0.447	13.05%
− threshold calibration	82.53	5.29	0.393	13.27%
− explicit fallback	82.02	5.15	0.340	17.98%
Random K (matched cost)	81.32	5.41	0.402	15.30%
Oracle $K^\star$	82.92	4.54	0.321	11.58%

Table 8: Normalized teacher–student distortion across latent depth, averaged over all eight benchmarks.

K	Full	− monotonicity	− nested	− step supervision
1	1.00	1.00	1.06	1.08
2	0.70	0.82	0.86	0.91
4	0.47	0.68	0.71	0.76
8	0.30	0.58	0.56	0.62
16	0.21	0.49	0.44	0.50

Figure 5 complements the ablation results by showing teacher distortion across depth. Removing self-distillation decreases average accuracy by 1.16 points and increases Avg.K by 16.6%, supporting its role as a shared decision reference across exposed depths. Removing step supervision gives the largest representation-quality drop: average accuracy decreases by 1.49 points, Avg.K rises from 5.30 to 6.79, and unsafe error rises from 12.30% to 14.55%. Removing nested training has the largest depth effect: Avg.K rises to 7.58, a 43.0% increase, while relative GPU cost rises by 21.1%. Removing monotonicity changes final accuracy less strongly but increases Avg.K and unsafe error, consistent with its role in stable refinement.

The fixed-$K_{\max}$ control reaches essentially the same average accuracy as the full system but requires Avg.$K = 16.00$. Random matched-cost allocation keeps the average computation close to NEST while losing 1.36 accuracy points and increasing unsafe error to 15.30%. The oracle result reaches 82.92 average accuracy at Avg.$K = 4.54$, showing that the trained hierarchy contains low-cost valid endpoints that the learned execution policy does not always identify.

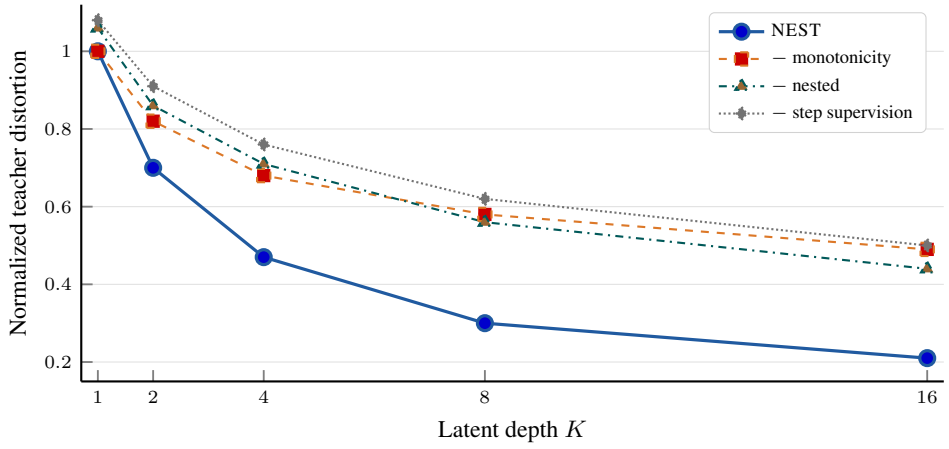


Figure 5: Teacher distortion across computation depth, averaged over all eight benchmarks. The full hierarchy exhibits the strongest progressive decrease.

Table 9: Measured latent-trajectory structure averaged over all eight benchmarks.

Variant	Mean Pairwise Cos↓	Effective Rank↑	Monotonic Pairs↑
CODI	0.79	4.8	0.70
SIM-CoT	0.64	7.1	0.82
NEST − step supervision	0.82	4.4	0.76
NEST − nested	0.70	6.0	0.83
NEST	0.56	8.8	0.92

D.2 Transfer and robustness

D.2.1 Depth-ladder sensitivity

The hierarchy remains stable across several candidate-depth sets. A denser ladder offers slightly finer allocation, while the five-point geometric ladder captures nearly all of the observed accuracy gain with fewer exposed readouts. This experiment tests whether the method depends on the exact values $\{1, 2, 4, 8, 16\}$ or on the broader training principle of exposing several nested endpoints.

Table 10 shows that the dense ladder improves eight-benchmark average accuracy by only 0.08 points relative to the five-depth ladder and reduces Avg.K from 5.30 to 5.02. The small accuracy difference indicates that endpoint competence is not tied to geometric spacing. The modest depth reduction is expected because a denser ladder gives the execution policy more stopping locations. We retain the five-depth ladder in the main experiments because it captures almost all of the quality gain while reducing the number of answer readouts that must be supervised and calibrated.

Table 10: Depth-ladder sensitivity averaged over all eight benchmarks.

Depth ladder	Avg. Acc.↑	Avg.K ↓	Rel.GPU↓	Unsafe↓
$\{1, 4, 8, 16\}$	82.50	5.61	0.412	12.65%
$\{2, 4, 8, 16\}$	82.56	5.54	0.410	12.55%
$\{1, 2, 4, 8, 16\}$	82.68	5.30	0.402	12.30%
$\{1, 2, 3, 4, 6, 8, 12, 16\}$	82.76	5.02	0.399	12.18%
$\{1, 4, 7, 10, 13, 16\}$	82.59	5.27	0.407	12.42%

D.2.2 Teacher-quality sensitivity

The teacher supplies structural supervision while ground-truth answer loss remains active at every exposed endpoint. Table 11 separates teacher answer quality from trace integrity. The main teacher is a frozen copy of the same Qwen3.5-4B checkpoint used to initialize the student; the weaker/same/stronger rows change only the teacher condition used to construct Stage-A supervision. Importantly, the teacher-accuracy diagnostic is computed on the *hierarchy-training trace-generation records*, macro-averaged over the eight benchmark training pools under the Stage-A trace-generation decoding configuration. It is therefore not a test-set Explicit-CoT baseline and should not be compared numerically with the Explicit CoT row in Table 1. In the two trace-integrity rows, the same-checkpoint Qwen3.5-4B teacher first generates its complete traces and final answers once; only the intermediate reasoning steps used for student-side process supervision are perturbed post hoc. Teacher final answers are therefore held fixed in those rows. A stronger teacher provides a modest gain; a weaker teacher degrades performance without collapsing the hierarchy. Corrupting or shuffling intermediate steps degrades the student despite unchanged trace-pool teacher answer accuracy, with shuffling producing the larger drop.

The teacher-accuracy column is a training-pool diagnostic rather than an alternate test-set baseline. Ground-truth answer supervision remains part of the student objective at every exposed endpoint, while the teacher functions as a structural reference for decision geometry and intermediate progression. A stronger teacher improves NEST modestly, while a weaker teacher reduces performance without destroying the hierarchy. These results indicate that the training procedure does not require an overwhelmingly stronger oracle.

Trace integrity matters independently of teacher answer quality. Because the 10% corruption and shuffled-order interventions are applied only after teacher generation, both retain the same 83.1% trace-pool teacher final-answer accuracy as the unperturbed same-checkpoint Qwen3.5-4B teacher.

Table 11: Teacher-quality and trace-integrity sensitivity over all eight benchmarks. Trace-pool Teacher Acc. is the frozen teacher's final-answer accuracy on the hierarchy-training records used for Stage-A trace generation, macro-averaged across benchmark training pools under the trace-generation decoding configuration; NEST Avg. Acc. is measured on the disjoint final test sets. The 83.1% teacher diagnostic is therefore not directly comparable to the Explicit-CoT test accuracy in Table 1. For the two "Same teacher" trace-integrity interventions, teacher final answers are held fixed and only intermediate reasoning steps used for process supervision are perturbed post hoc.

Teacher condition	Trace-pool Teacher Acc.	NEST Avg. Acc.	Avg.K	Unsafe
Weaker explicit teacher	79.1	81.77	5.79	13.70%
Same-checkpoint Qwen3.5-4B teacher	83.1	82.68	5.30	12.30%
Stronger teacher	86.5	83.10	5.12	11.80%
Same teacher + 10% corrupted steps	83.1	81.94	5.67	13.70%
Same teacher + shuffled step order	83.1	81.56	6.01	14.10%

Nevertheless, corrupting intermediate steps lowers NEST accuracy from 82.68 to 81.94 and raises Avg.K from 5.30 to 5.67, while shuffling the same steps lowers NEST accuracy further to 81.56 and raises Avg.K to 6.01. The result isolates the value of coherent ordered process information rather than conflating trace quality with teacher answer quality.

Table 12: Three-run cross-backbone results averaged equally over all eight benchmarks. Accuracy is mean±std.

Backbone	Method	Avg. Acc.↑	Rel.GPU↓
R1-Distill-Qwen-7B	Explicit CoT	87.42 ± 0.19	1.000 ± 0.011
	SwiReasoning	87.78 ± 0.21	0.629 ± 0.014
	NEST	88.25 ± 0.18	0.439 ± 0.007
R1-Distill-Llama-8B	Explicit CoT	85.36 ± 0.22	1.000 ± 0.012
	SwiReasoning	85.69 ± 0.24	0.628 ± 0.015
	NEST	86.58 ± 0.21	0.459 ± 0.007

Cross-backbone eight-benchmark results Table 12 shows that the transfer result persists when accuracy is averaged over the same eight-benchmark suite used in the primary evaluation. NEST improves over SwiReasoning on both distilled backbones while using substantially less relative GPU cost.

Cross-backbone results preserve the same quality–cost direction. On R1-Distill-Qwen-7B, eight-benchmark average accuracy rises from 87.78 for SwiReasoning to 88.25 for NEST, while Rel.GPU falls from 0.629 to 0.439. On R1-Distill-Llama-8B, average accuracy rises from 85.69 to 86.58 and Rel.GPU falls from 0.628 to 0.459. The OpenMathInstruct-1 scaling study below uses the same eight-benchmark evaluation and shows a smooth accuracy increase as additional training pairs are added.

OpenMathInstruct-1 scaling block The following scaling results are obtained when additional OpenMathInstruct-1 pairs are added (Toshniwal et al., 2024). Each operating point is repeated across three seeds.

Table 13: Three-run OpenMathInstruct-1 scaling results averaged equally over all eight benchmarks. The zero-extra-data row uses the same base NEST configuration; its standard deviation is computed over the scaling-study seed set.

Training regime	Extra pairs	Avg. Acc.↑	Avg.K ↓	Rel.GPU↓
Base NEST training	0	82.68 ± 0.17	5.30 ± 0.18	0.402 ± 0.012
+ small	100k	82.87 ± 0.21	5.19 ± 0.14	0.398 ± 0.011
+ medium	500k	83.19 ± 0.16	5.03 ± 0.19	0.394 ± 0.009
+ large	1.8M	83.42 ± 0.18	4.94 ± 0.13	0.391 ± 0.010

Table 13 shows an eight-benchmark trend: adding 1.8M pairs raises average accuracy from 82.68 to 83.42 while reducing Avg.K from 5.30 to 4.94. The repeated-run variability remains small across scaling levels, supporting a stable trend within the measured range.

D.3 ADAPTIVE EXECUTION AND CALIBRATION

Pool-free deployment check. The deployed policy never queries the offline candidate set: pre-decode confidence uses only the vocabulary statistics in Eqs. 8–9 and latent-state dynamics, and the full answer is greedily decoded only after acceptance. This pool-free implementation reaches 82.58 average accuracy, Rel.GPU 0.410, and 12.45% unsafe latent error. Hierarchy training, router/confidence fitting, threshold calibration, and final testing are also disjoint; the calibration split never updates the confidence head. Under this separate strict-split deployment check, NEST reaches 82.60 average accuracy, Avg.$K = 5.40$, Rel.GPU 0.410, Unsafe 12.40%, and ECE 0.020; these deployment-only numbers are not used as the primary operating point elsewhere. Split definitions, exact inference pseudocode, oracle headroom, risk–coverage, and task-shift analyses are in Appendices B and D.

Table 14: Distribution of the minimal sufficient depth $K^\star$. Outcome fractions are computed within each benchmark and then averaged equally across the eight benchmarks, matching the aggregation used for Oracle Avg.K.

Outcome	Fraction
$K^\star = 1$	15.8%
$K^\star = 2$	23.9%
$K^\star = 4$	21.8%
$K^\star = 8$	17.1%
$K^\star = 16$	10.4%
Explicit fallback	11.0%

Minimal-depth distribution Table 14 summarizes computation demand directly: 61.5% of inputs have a sufficient endpoint by $K \leq 4$. Because the $K^\star$ fractions are macro-averaged with the same benchmark weights as Oracle Avg.K and immediate fallback executes zero latent updates, $1(0.158) + 2(0.239) + 4(0.218) + 8(0.171) + 16(0.104) = 4.54$, matching the oracle mean reported elsewhere. The capacity predictor exactly matches the oracle ladder position on 68.4% of inputs, lies within one adjacent level on 91.7%, under-allocates on 11.8%, and over-allocates on 19.8%. Mean absolute ladder-index error is 0.43. The initial mean predicted depth is 5.12, the executed mean after confidence escalation is 5.30, and the oracle mean is 4.54.

Table 15: Conditional latent error along the eight-benchmark NEST risk–coverage curve under the equal-weight benchmark mixture used for the main routing metrics.

Coverage (%)	70	75	80	85	85.65	89.5	90	95
Cond. error (%)	7.8	8.9	10.7	13.8	14.36	14.83	15.0	19.2

Risk–coverage operating points Table 15 makes the coverage–risk trade-off explicit: conditional error increases as the acceptance threshold is relaxed from 70% to 95% coverage. The calibrated operating point is $(85.65\%, 14.36\%)$, while the no-threshold-calibration operating point is $(89.5\%, 14.83\%)$; both are points on the same score-induced risk–coverage curve. Score-level calibration and ranking metrics are summarized separately in Table 16.

The capacity predictor exactly matches the oracle ladder position on 68.4% of inputs and falls within one adjacent level on 91.7%. Under-allocation is 11.8%, over-allocation is 19.8%, and the mean absolute ladder-index error is 0.43. Confidence escalation changes the mean initial prediction of 5.12 to an executed mean of 5.30, partially repairing under-allocation.

Figure 6 shows the corresponding risk–coverage curve. Held-out threshold calibration trades a small amount of coverage for lower unsafe acceptance. Relative to the uncalibrated operating point, latent coverage changes from 89.5% to 85.65%, while unsafe latent error falls from 13.27% to 12.30%. Because threshold selection does not update the confidence head or the underlying score, ECE, Brier, AUROC, and full-curve AURC remain unchanged. Across the sampled thresholds, the risk–coverage curve is nondecreasing as the system accepts progressively harder predictions.

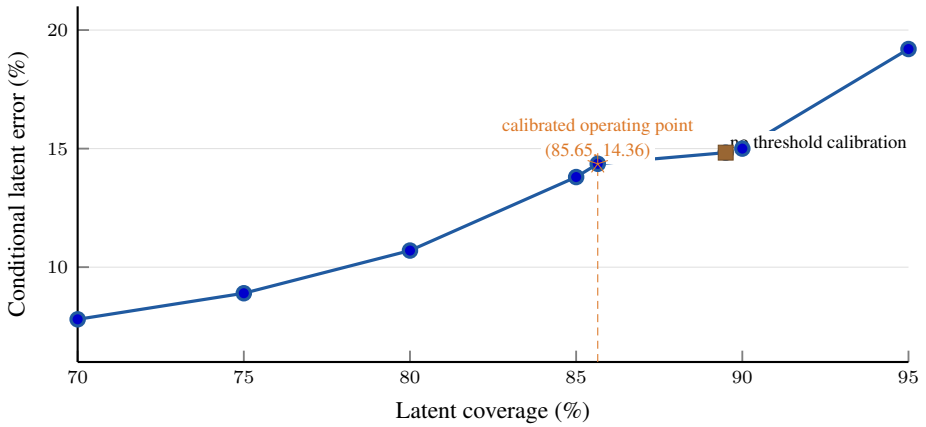


Figure 6: Eight-benchmark risk–coverage curve for NEST.

Calibration summary Table 16 evaluates the quality of the acceptance score separately from the raw answer model. NEST improves ECE, Brier score, correctness AUROC, and AURC relative to SwiReasoning, indicating better confidence quality and selective-prediction behavior.

The no-threshold-calibration row uses the same trained confidence score as NEST and differs only in the selected acceptance threshold. Accordingly, ECE, Brier, AUROC, and full-curve AURC are unchanged, while coverage, unsafe error, and conditional error move to a different operating point. This isolates the role of held-out threshold selection from the quality of the underlying score.

Table 16: Eight-benchmark calibration and selective-prediction summary. Fallback, coverage, and unsafe error use the same equal-weight benchmark mixture as Table 1; conditional error is the ratio Unsafe/Coverage under that mixture. These quantities are fractions here, whereas Table 1 reports fallback and unsafe error as percentages.

Method	Fallback	Coverage	Unsafe↓	Cond.Err.↓	ECE↓	Brier↓	AUROC↑	AURC↓
SwiReasoning	.1302	.8698	.1324	.1522	.032	.055	.86	.079
Entropy-only	.1500	.8500	.1262	.1485	.043	.067	.82	.086
NEST − threshold calibration	.1050	.8950	.1327	.1483	.016	.038	.91	.067
NEST	.1435	.8565	.1230	.1436	.016	.038	.91	.067
Oracle $K^\star$	.1100	.8900	.1158	.1301	.012	.031	.94	.056

D.3.2 Calibration under task shift

We freeze the source-calibrated threshold and evaluate it under held-out task shift. Reliability degrades, then recovers with a small target calibration set. This separates transfer of the latent hierarchy from transfer of the acceptance threshold. The experiment is deliberately evaluated without retuning the source threshold first, because that setting exposes calibration shift directly.

Table 17 reports threshold-calibration transfer after aggregating each setting equally over all eight benchmarks. Under the frozen source threshold, unsafe error rises from 12.30% to 14.70% and conditional latent error rises from 14.36% to 17.95%. Adding only 200 target-domain calibration examples recovers a substantial fraction of the loss, and 500 examples bring unsafe error to 12.80%. The hierarchy therefore remains usable under task shift, while the acceptance threshold benefits from domain-specific recalibration when the confidence distribution changes.

D.3.3 Capacity-predictor ablation

Table 18 explains why the predictor combines semantic and uncertainty signals. Question embeddings alone under-allocate 17.2% of inputs. Direct-answer entropy and top-two margin reduce this rate before any latent state is computed. The confidence stage then observes trajectory dynamics and raises the executed mean from 5.12 to 5.30, repairing part of the remaining under-allocation. The

Table 17: Eight-benchmark threshold-calibration transfer under task shift. Each row uses the same equal-weight benchmark mixture as the main routing metrics; conditional error is Unsafe/Coverage under that mixture.

Calibration setting	Coverage	Unsafe↓	Cond.Err.↓
In-domain calibration	85.65%	12.30%	14.36%
Frozen threshold under shift	81.9%	14.70%	17.95%
+ 200 target calibration examples	82.6%	13.60%	16.46%
+ 500 target calibration examples	83.5%	12.80%	15.33%

Table 18: Capacity-predictor feature ablation pooled over all eight benchmarks.

Predictor	Exact match	Under-alloc.	Initial Avg.K	Final Avg.K
Question embedding only	60.9%	17.2%	4.81	5.63
+ direct-answer entropy	64.5%	14.8%	4.95	5.47
+ top-2 margin	66.7%	13.3%	5.03	5.39
Full predictor	68.4%	11.8%	5.12	5.30
Oracle	100%	0%	4.54	4.54

oracle mean of 4.54 shows that future gains can come from better budget prediction without changing the learned hierarchy.

D.4 Efficiency and stress tests

D.4.1 Measured training cost

Table 19 reports the measured one-time model-training costs for teacher-trace generation, hierarchy training, execution-head fitting, and calibration. The offline open-ended candidate-pool job is training-side preprocessing and is not included in these measured GPU-hour totals; we state this explicitly rather than treating an unmeasured preprocessing cost as zero. It has zero deployed test-time generation cost because $\mathcal{C}(x)$ is neither built nor queried by the inference router.

Table 19: Measured model-training overhead on NVIDIA GeForce RTX 4090 GPUs. Offline candidate-pool preprocessing is not included in the GPU-hour totals.

Component	GPU-hours
Teacher trace generation	3.8
Terminal latent baseline training	20.9
NEST hierarchy training	26.4
Capacity predictor	0.7
Confidence head	0.4
Calibration	0.1
NEST total	31.4
Comparable Explicit-CoT pipeline	18.6
Extra NEST overhead	12.8

The measured model-training total is 31.4 GPU-hours for NEST versus 18.6 GPU-hours for the comparable Explicit-CoT pipeline, corresponding to 12.8 additional GPU-hours. Offline candidate-pool preprocessing is not included in Table 19; it is a training-side preprocessing cost and is absent from deployed inference.

Absolute-efficiency measurements Table 20 reports absolute efficiency under the shared inference configuration for the eight-benchmark evaluation suite. NEST requires 430 ms/query, 0.402 relative GPU-seconds/query, and 14.4 GB peak memory, compared with 1000 ms/query, 1.004 relative GPU-seconds/query, and 20.0 GB for Explicit CoT. Rel.GPU and end-to-end latency are measured independently by the shared harness; the redundant latency-ratio column is omitted because it is only a normalization of the millisecond measurements.

Rel.GPU is normalized from accelerator-active time, whereas latency includes the full host-to-device request path and therefore also contains approximately fixed runtime overhead. With the 1000-ms Explicit-CoT latency reference, the intercept corresponds to about 30 ms/query of approximately fixed end-to-end overhead.

Table 20: Measured absolute inference efficiency on NVIDIA GeForce RTX 4090 under the shared configuration, averaged over all eight benchmarks. Rel.GPU uses accelerator-active timing, whereas latency is timed independently around the complete request path.

Method	Rel.GPU	Latency (ms)	Peak mem. (GB)
Explicit CoT	1.004	1000 ± 35	20.0
SIM-CoT	0.451	480 ± 22	15.6
SwiReasoning	0.592	620 ± 27	16.4
NEST	0.402	430 ± 20	14.4
Oracle $K^\star$	0.321	350 ± 18	13.6

SwiReasoning is the strongest adaptive baseline in the primary block. Across the full eight-benchmark suite, NEST improves average accuracy from 82.07 to 82.68, reduces Avg.K from 6.44 to 5.30, reduces relative GPU cost from 0.592 to 0.402, and reduces unsafe latent error from 13.24% to 12.30%. At the individual-task level, NEST improves on seven of the eight benchmarks and remains competitive on GSM8K, so the efficiency advantage is not obtained by sacrificing broad benchmark coverage.

E Experimental Protocol and Reproducibility

E.1 Datasets, models, and evaluation regimes

The evaluation spans eight benchmark families. GSM8K (Cobbe et al., 2021), MATH-500 (Hendrycks et al., 2021), SVAMP (Patel et al., 2021), BBH (Suzgun et al., 2023), MultiArith (Roy & Roth, 2015), AQuA-RAT (Ling et al., 2017), PrOntoQA (Saparov & He, 2023), and ProsQA (Hao et al., 2024) cover complementary reasoning regimes. The primary model is Qwen3.5-4B (Qwen Team, 2026); cross-backbone evaluation uses DeepSeek-R1-Distill-Qwen-7B and DeepSeek-R1-Distill-Llama-8B (DeepSeek-AI, 2025). OpenMathInstruct-1 (Toshniwal et al., 2024) is used for the training-data scaling study. In the primary self-distillation setting, teacher and student use the same Qwen3.5-4B base checkpoint: the teacher copy is frozen and generates explicit traces/targets, while the student copy is the model optimized and deployed.

GSM8K supplies grade-school arithmetic reasoning; MATH-500 emphasizes harder competition mathematics; SVAMP tests robustness under systematic word-problem variation; BBH covers heterogeneous multi-step reasoning; MultiArith and AQuA-RAT test arithmetic and quantitative transfer; PrOntoQA and ProsQA extend the evaluation toward deductive and search-oriented reasoning. The same answer-normalization and evaluation parser is used across compared methods on each benchmark. For MATH-500, each of the five primary runs evaluates all 500 test questions. Repeated-run means and sample standard deviations are computed from the per-run outputs before the display rounding used in Table 1.

E.2 Baseline families and fairness constraints

The comparison set is organized into direct prediction, explicit reasoning, continuous latent reasoning, and adaptive execution. Direct Answer predicts the final answer directly, without generating an explicit reasoning trace or running recurrent latent reasoning. Explicit CoT generates an explicit textual chain of thought before the final answer and serves as the explicit-reasoning compute reference. Coconut recurrently feeds continuous hidden states back into the model as latent reasoning states (Hao et al., 2024). CODI transfers explicit reasoning behavior into implicit computation through self-distillation (Shen et al., 2025). SIM-CoT uses step-level supervision to train longer implicit reasoning trajectories (Wei et al., 2026). SwiReasoning is the strong adaptive latent/explicit baseline, using confidence-based adaptive computation to decide how reasoning should proceed (Shi et al., 2026). Fixed-K latent reasoning executes a predetermined number of recurrent latent updates for every input and therefore provides a non-adaptive continuous-reasoning control. The Random K matched-cost row used in the ablation assigns computation without the learned allocation policy while matching average cost, and Oracle $K^\star$ uses the measured minimal sufficient depth as an upper-bound control rather than a deployable baseline.

Fairness constraints include shared benchmark splits, answer normalization, evaluation parser, decoding settings, maximum answer length, hardware/inference harness, batching regime, warm-up procedure, cache settings, and random-seed sets whenever the corresponding baseline implementation permits them. Latency comparisons use the same caching and batching regime; self-consistency or multi-sample decoding is not compared directly against greedy decoding without counting every generated sample.

Training-side auxiliary supervision is not identical across heterogeneous method families, so Table 1 remains an end-to-end system comparison rather than a causal objective comparison. We therefore add a dedicated matched data/teacher/compute study for the strongest adaptive baseline. Both SwiReasoning and NEST use the same core examples, backbone, teacher checkpoint, question-only teacher-trace pool, answer supervision, optimizer-step budget, and measured training-compute budget. Auxiliary targets remain method-specific: NEST consumes its candidate-distribution and hidden-state alignment targets, whereas SwiReasoning uses the targets required by its own formulation. We therefore do not call this a fully matched-supervision ablation. Under this control, SwiReasoning reaches 82.35 ± 0.18 average accuracy with Avg.$K = 6.25$, Rel.GPU 0.565, and Unsafe 12.85%, while NEST reaches 82.68 ± 0.16, 5.30, 0.402, and 12.30%. Table 21 records the comparison. Causal attribution inside NEST still comes from the terminal-only, Core-NEST, and full-objective controls, which share the same backbone, router capacity, data, and calibration protocol.

For the eight-benchmark cross-framework intervention in Table 3, the original and +Core NEST variants of each host framework are fitted with the same generic capacity/confidence execution layer on the same router split and calibrated on the same held-out calibration split. Host architecture, execution-policy capacity, and stopping protocol are therefore fixed within each pair; only multi-depth endpoint supervision changes.

Table 21: Five-run matched data/teacher/compute comparison over all eight benchmarks with the strongest adaptive baseline. Training examples, backbone, teacher checkpoint, question-only trace pool, answer supervision, optimizer-step budget, and training-compute budget are controlled; auxiliary targets remain method-specific. Accuracy is mean±std.

Method	Avg. Acc.↑	Avg.K ↓	Rel.GPU↓	Unsafe↓
SwiReasoning (controlled)	82.35 ± 0.18	6.25	0.565	12.85%
NEST (controlled)	82.68 ± 0.16	5.30	0.402	12.30%

E.3 Hyperparameters and released run configuration

Table 22 collects the run configuration.

E.4 Data separation and leakage control

We use four disjoint data roles after the benchmark's official test set is separated: hierarchy training, router/confidence fitting, threshold calibration, and final testing. Candidate preprocessing is an offline label-stripped view of records used only to construct supervision artifacts; it is not an additional source of labels. The split roles are summarized in Table 23.

Candidate generation receives a physically label-stripped view containing only example identifiers and questions; its API has no gold-answer argument. Gold answers may supervise endpoint CE and router-fit correctness labels in their designated splits, but never candidate generation, candidate filtering/ranking, candidate-pool hyperparameters, or test-time execution. The calibration split is never used to update the capacity or confidence heads. Test-time inference never touches $\mathcal{C}(x)$, teacher traces/logits/states, gold answers, or oracle $K^\star$.

A reproducible implementation therefore asserts that `teacher_trace`, `teacher_logits`, `gold_answer`, `oracle_K`, and cached candidate artifacts are absent from test-time capacity/confidence inputs. Pipeline label independence should not be confused with the broader and generally unverifiable claim that a pretrained generator has never encountered a public benchmark during pretraining; we make only the former claim.

Table 22: Main-run hyperparameter checklist.

Setting	Main configuration
Teacher / student checkpoint	Same Qwen3.5-4B base checkpoint for the primary runs; the teacher is a frozen copy used only to construct training-side traces/targets, while the student copy is optimized and deployed
Student depth ladder	$\{1, 2, 4, 8, 16\}$
Depth-view distribution P_{depth}	Uniform over all 3-element subsets of $\{1, 2, 4, 8, 16\}$, with $P(v) = \frac{1}{\binom{5}{3}} = \frac{1}{10}$ for each view; losses are averaged uniformly within a view
Optimizer	AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$ and $\epsilon = 10^{-8}$
Peak learning rate / schedule	2×10^{-5} for the backbone and 1×10^{-4} for the teacher projection; cosine decay
Warm-up	3% of optimizer steps
Weight decay	0.01
Effective batch size / grad. accumulation	64 / 64 with microbatch size 1
Training steps or epochs	3 epochs; $3\,\lceil N_{\text{hierarchy}}/64 \rceil$ optimizer steps
Gradient clipping	1.0
$\lambda_{\text{ans}}, \lambda_{\text{KD}}, \lambda_{\text{step}}, \lambda_{\text{mono}}$	1, 1, $\frac{1}{5120}$, $\frac{1}{10}$
Distillation temperature T_{KD} / routing slack ξ	2.0 / 0.01
Candidate beams	8
Nucleus samples	8 per temperature
Candidate temperatures	$\{0.4, 0.7, 1.0\}$
Candidate top-p	0.95
Candidate max new tokens	512
Max unique candidates	32
Sequence length exponent α	1.0
Candidate-generation seed	1234
Primary repeated runs	5
Cross-backbone/scaling repeated runs	3
Accelerator	NVIDIA GeForce RTX 4090 (24 GB)

Table 23: Disjoint data roles. "Allowed" lists fields that may influence fitted parameters or threshold choice in that stage.

Role	Allowed information	Updates
Hierarchy train	x, gold answer for endpoint CE, question-only teacher trace, cached candidate scores/states	student hierarchy θ
Router fit	x, frozen-hierarchy states, offline $K^\star$, endpoint correctness labels	q_ψ, g_ω
Calibration	frozen confidence scores and correctness labels	threshold $\tau^\star$ only
Test	x and student-generated states/logits	none

E.5 Metrics and how to interpret them

Task quality. Accuracy (or exact match when the benchmark uses normalized string matching) is the fraction of evaluated questions whose final system answer is correct. For repeated runs, we report mean and standard deviation across seeds. Unless a table explicitly states otherwise, eight-benchmark aggregate metrics use an equal-weight benchmark mixture: the relevant per-benchmark quantity is computed first and then averaged across the eight benchmarks. For joint routing probabilities, this is equivalent to sampling a benchmark uniformly and then a query uniformly from that benchmark. Aggregate conditional latent error is reported as aggregate Unsafe divided by aggregate Coverage under this same mixture, rather than as an unweighted mean of eight per-benchmark conditional-error ratios. This convention makes the routing quantities in Tables 1, 16, and 17 directly comparable.

Amount of latent computation. For query i, let K_i be the number of recurrent latent updates actually executed before acceptance or fallback. Then

$$\text{Avg.}K = \frac{1}{N} \sum_{i=1}^{N} K_i. \tag{12}$$

This includes escalation from an initial predicted depth to deeper candidate depths. Lower Avg.K means that the system reaches accepted endpoints earlier on average; it is not treated as a direct FLOP count.

Measured compute and memory. All reported hardware measurements use the same NVIDIA GeForce RTX 4090 (24 GB) device type. GPU-seconds/query is measured from accelerator-active timing under the shared harness, and Rel.GPU is that quantity normalized to the Explicit-CoT reference; a value of 0.402 means 40.2% of the reference accelerator time. End-to-end latency is measured independently around the complete request path, so it additionally includes host/runtime work that is outside the accelerator-active timer. Peak GPU memory is the maximum allocated device memory, and throughput is completed queries per second. All timing measurements include any fallback triggered at the reported operating point.

Coverage and fallback. Fallback is the fraction of queries that invoke explicit CoT. Latent Coverage is the complementary fraction,

$$\text{Coverage} = 1 - \text{Fallback}. \tag{13}$$

High coverage means that more requests are completed by the latent path; it is useful only together with an error measure.

Unsafe and conditional latent error. Unsafe latent error is the joint system-level probability

$$R_{\text{unsafe}} = P(\text{latent accepted and wrong}). \tag{14}$$

Conditional latent error measures the error rate among accepted latent predictions,

$$R_{\text{cond}} = P(\text{wrong} \mid \text{latent accepted}). \tag{15}$$

The first answers "how often does the deployed latent path create a wrong accepted response among all requests?"; the second answers "given that the latent path accepted, how often was it wrong?"

Calibration metrics. For confidence score $c_i \in [0,1]$ and correctness label $b_i \in \{0,1\}$, the Brier score is $N^{-1}\sum_i (c_i - b_i)^2$. ECE partitions predictions into confidence bins and computes a weighted average of the absolute gap between mean confidence and empirical accuracy in each bin. AUROC measures how well confidence ranks correct predictions above incorrect ones and is threshold independent. A risk–coverage curve varies the acceptance threshold and plots conditional error against accepted coverage; AURC is the area under this curve. Lower ECE, Brier, and AURC are better, while higher AUROC is better.

Trajectory diagnostics. Teacher distortion $D_K = \|\mathbf{z}_K^{\mathrm{S}} - \mathbf{z}^{\mathrm{T}}\|_\infty$ measures endpoint disagreement with the common teacher decision reference in the shared finite candidate-set log-probability space. Adjacent-depth monotonicity is the fraction of evaluated transitions satisfying $D_{K_{j+1}} \leq D_{K_j}$. Pairwise cosine similarity measures redundancy among latent states, with lower values indicating more directional diversity. Effective rank summarizes how many independent directions carry substantial variance in the state matrix; higher values indicate a less collapsed trajectory. Fixed-prefix accuracy directly measures whether a chosen depth can solve the task without help from deeper states.