

Latent and continuous reasoning

Token feedback, hidden-state feedback and recurrent depth.

Reasoning agents, feedback, and scientific evidence

Must every intermediate step become a token?

A model already performs internal vector computation, yet an ordinary reasoning trace repeatedly turns that state into a token and then an embedding.

Text makes intermediate work readable and trainable from demonstrations. It also commits computation to a particular vocabulary interface.

Could some intermediate work use another interface? This question motivates an architectural experiment, not a conclusion that text is the cause of reasoning failures.

The hidden-state interface creates a possibility

Inspect the existing loop: hidden state, vocabulary prediction, chosen token, input embedding. The hidden state is already available before token selection.

Feeding that state back suggests a continuous alternative. Compressed text or explicit branches are also plausible responses, with different supervision and checking properties.

A new interface creates a training problem: what makes its states useful for the answer? Coconut uses a staged curriculum to connect textual reasoning and continuous feedback.

Choosing an intermediate representation

Candidate	Reason to choose it	Cost or limitation
Text or structured tokens	Readable steps and familiar supervision	Vocabulary choices may constrain the interface
Explicit branches	Keep inspectable alternative trajectories	Expansion and selection consume computation
Continuous feedback	Reuse hidden states between reasoning steps	Needs training and independent quality checks

Continuous feedback is attractive if useful computation survives the interface change. Test matched resources rather than infer efficiency from token counts.

Pedagogical reconstruction

[Hao et al. \(2024\)](#)

Coconut changes the feedback path

$$\begin{aligned} \text{Token : } & x_{t+1} = \text{Embed}(\text{token}_t) \\ \text{Continuous : } & x_{t+1} = z_t \end{aligned}$$

Coconut (Chain of Continuous Thought; Hao et al., 2024) feeds the final hidden state z_t directly into the next input embedding during a continuous phase, bypassing token selection.

The recurrent use of the language model remains computational work. Count latent steps and answer decoding alongside any token savings.

A training curriculum that replaces explicit reasoning steps with continuous ones helps define what the model learns to store.

Recurrent depth reuses parameters over computation

Geiping et al. (2025) study recurrent depth: a shared block repeatedly updates an internal state while reading the input. More iterations add computation without adding a new parameter set for each iteration.

Parameter count, unrolled compute, and latency are different quantities. Extra iterations may eventually provide diminishing gains or unstable states.

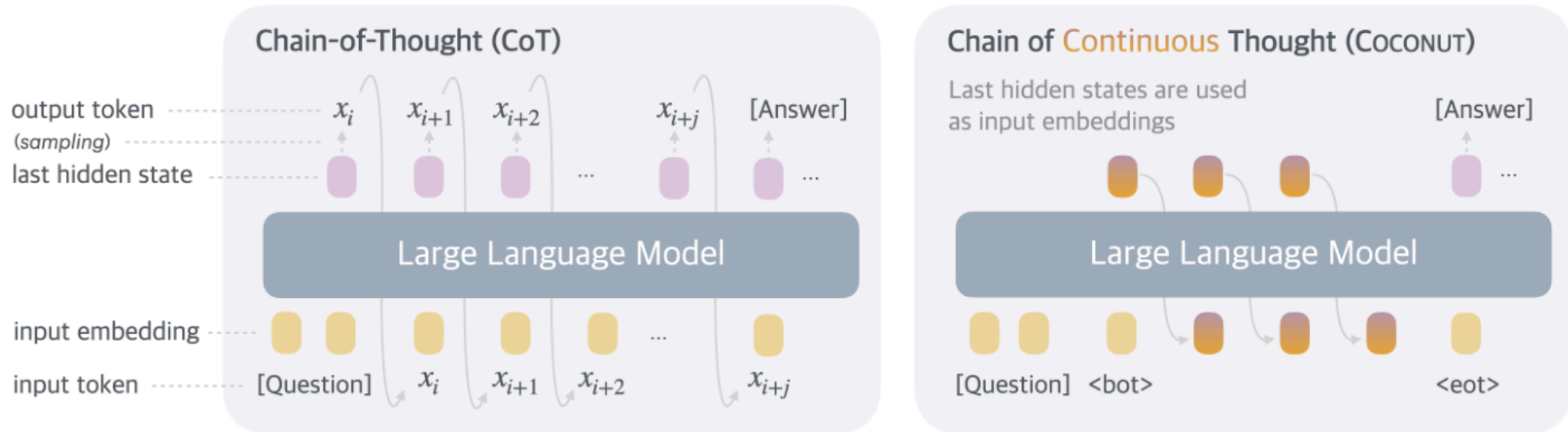
Evaluate the entire accuracy–compute curve, including the budget at which a non-recurrent baseline becomes preferable.

Coconut: what is fed back at each step?

Phase	Input to the next position	Output used next
Text reasoning	Embedding of the generated token	A discrete token sampled from logits
Enter latent mode	Context including the start marker	Last normalized hidden state
Continuous step	Previous hidden state as input embedding	Another hidden state; no token selection
Answer mode	End marker and accumulated context	Ordinary answer-token decoding

Coconut changes the feedback representation. Latent steps are still sequential forward computation, not free extra thought.

Coconut: token and hidden-state feedback



The token-generation and continuous-feedback paths differ in what becomes the next input. Hidden-state updates consume computation even when they produce no visible word.

Coconut learns through a staged curriculum

Curriculum stage k	Training input	Supervised target
$k = 0$	Question + textual reasoning	Remaining reasoning and answer tokens
$k > 0$	First k steps $\rightarrow k \times c$ latent states	Predict remaining text and answer
Latent segment	Run the differentiable feedback chain	Mask language-model loss at latent positions
Following text	Predict remaining text and answer	Backpropagate through the latent computation

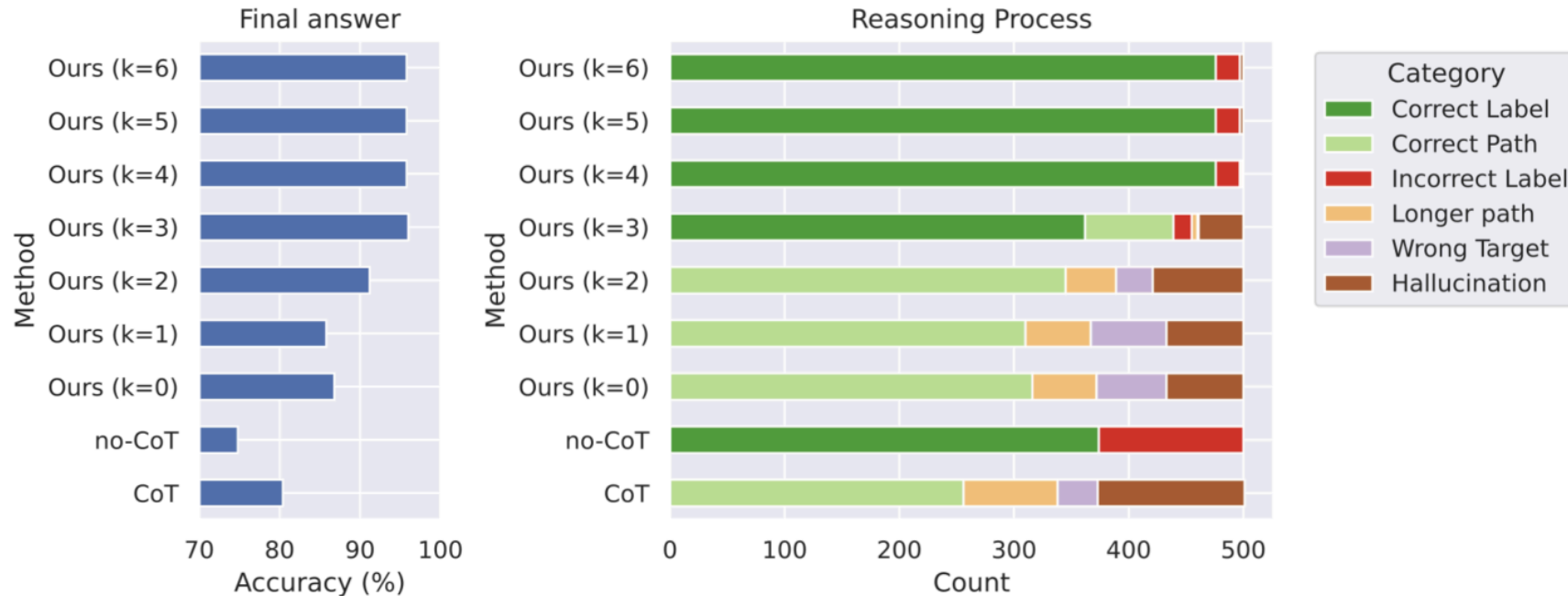
c is the number of latent positions per replaced reasoning step. The supervision comes through later tokens, not vector-level text matching.

Evidence: a gain on one task can reverse on another

Method	GSM8k		ProntoQA		ProsQA	
	Acc. (%)	# Tokens	Acc. (%)	# Tokens	Acc. (%)	# Tokens
CoT	42.9 $\pm$ 0.2	25.0	98.8 $\pm$ 0.8	92.5	77.5 $\pm$ 1.9	49.4
No-CoT	16.5 $\pm$ 0.5	2.2	93.8 $\pm$ 0.7	3.0	76.7 $\pm$ 1.0	8.2
iCoT	30.0*	2.2	99.8 $\pm$ 0.3	3.0	98.2 $\pm$ 0.3	8.2
Pause Token	16.4 $\pm$ 1.8	2.2	77.7 $\pm$ 21.0	3.0	75.9 $\pm$ 0.7	8.2
COCONUT (Ours)	34.1 $\pm$ 1.5	8.2	99.8 $\pm$ 0.2	9.0	97.0 $\pm$ 0.3	14.2
- <i>w/o curriculum</i>	14.4 $\pm$ 0.8	8.2	52.4 $\pm$ 0.4	9.0	76.1 $\pm$ 0.2	14.2
- <i>w/o thought</i>	21.6 $\pm$ 0.5	2.3	99.9 $\pm$ 0.1	3.0	95.5 $\pm$ 1.1	8.2
- <i>pause as thought</i>	24.1 $\pm$ 0.7	2.2	100.0 $\pm$ 0.1	3.0	96.6 $\pm$ 0.8	8.2

Coconut reaches 97.0% on ProsQA versus 77.5% for CoT, while GSM8K is 34.1% versus 42.9%. The table also reports token counts and curriculum ablations. This task dependence limits any universal superiority claim. Token count is an interface-level cost measure that omits other model computation.

Coconut continuous thoughts: interpretation and limits



Benefits vary across tasks and training conditions. A decoded trace or probe may not faithfully represent internal computation, and a reduction in visible tokens is not necessarily a proportional reduction in FLOPs or latency.

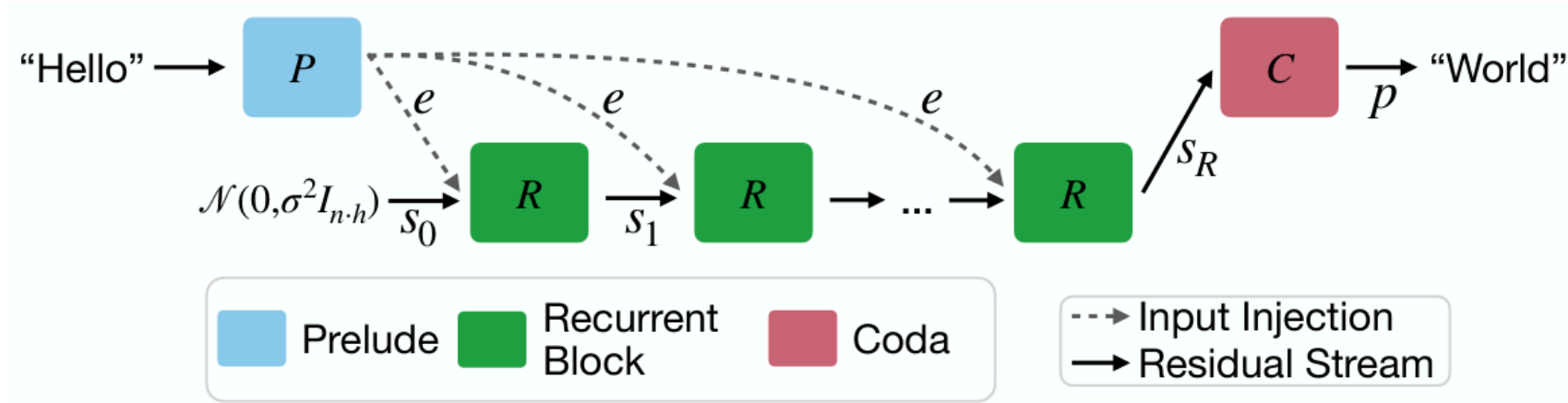
Recurrent-depth language models: the method

The architecture separates input processing, a parameter-sharing recurrent core, and output processing. The core can be iterated more times at inference.

This allows computation to increase through depth without allocating a distinct parameter set to every iteration.

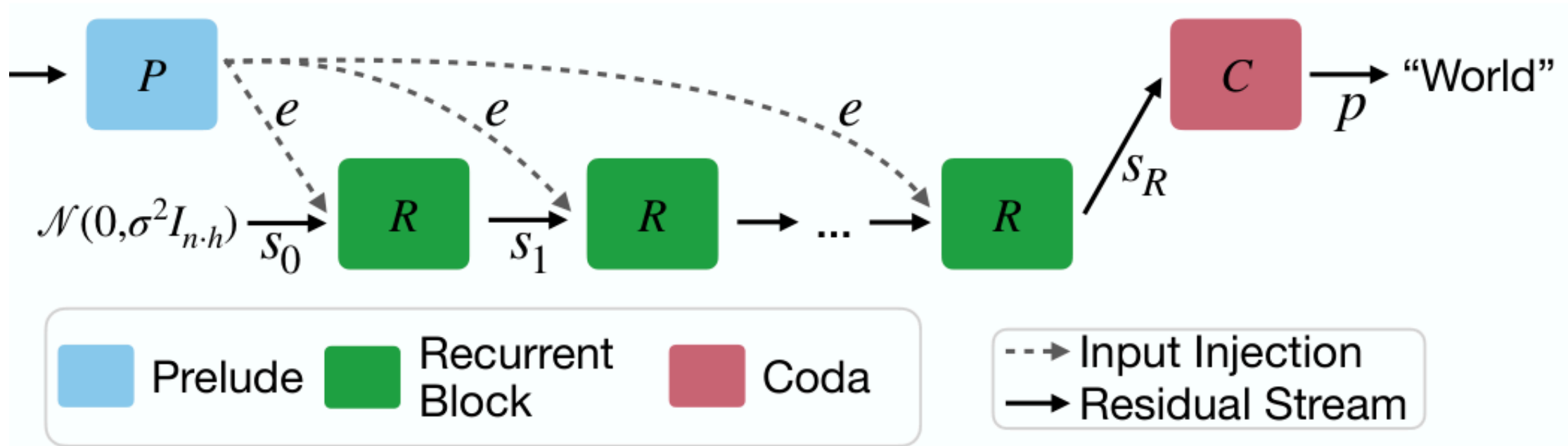
Training across recurrence counts and evaluating longer inference trajectories are central to the design. The iteration count is an additional control variable, not a guarantee of monotonic quality improvement.

Recurrent depth: shared-block architecture



A prelude feeds repeated applications of the recurrent block, followed by the coda. Input injection and residual state are distinct pathways. Increasing the iteration budget repeats computation while reusing parameters.

Recurrent-depth language models: mechanism detail



Each recurrent update changes the state while reusing parameters. Parameter sharing can reduce model storage without eliminating the inference computation required by repeated updates.

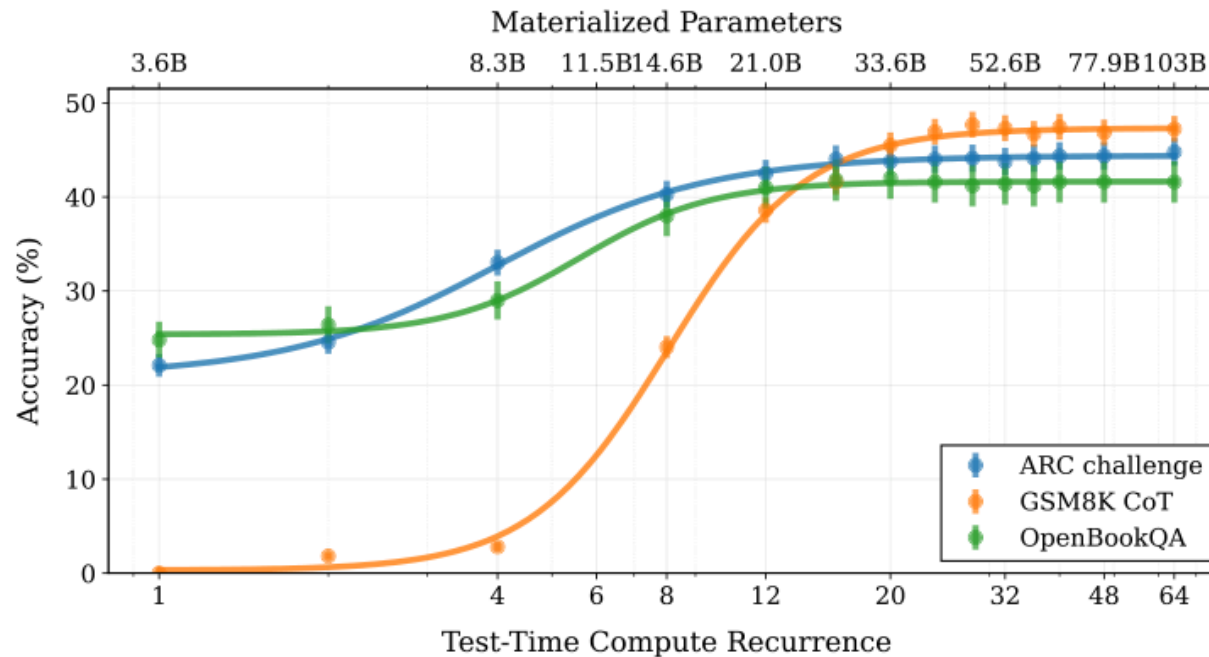
Evaluation scope

The original scaling figure plots accuracy against inference recurrence and an associated materialised-computation measure. Different tasks show different gains and saturation patterns.

Parameter count and unrolled computation are not interchangeable.

The curve metric and total inference resources determine whether a recurrent model supports a comparison with a larger non-recurrent baseline.

Recurrent-depth language models: interpretation and limits



Longer recurrence can saturate, destabilise, or alter latency. A converged hidden state can still produce a wrong answer. Scientific transfer and causal interpretation of intermediate states remain separate evaluation questions.

Quiet-STaR is not the same representation change

Quiet-STaR samples textual rationales around ordinary text positions and trains those rationales using their contribution to future-token prediction. The thoughts are explicit token sequences during this procedure.

Its relevance is the learning signal for intermediate reasoning, not proof that continuous hidden states are necessary.

Compare supervision, representation, and inference policy separately; grouping all forms of internal reasoning together obscures the causal source of a gain.

A latent is not automatically a probability distribution

A hidden vector may encode useful alternatives without assigning normalised, calibrated probabilities to them. A linear probe can reveal decodable information but does not prove the model uses it.

Interventions, controls, and downstream behavioural effects strengthen an interpretability claim.

Scientific decision-making needs reliable predictions or actions; it does not logically require a human-readable decode of every intermediate state. The public claim still needs independent evidential support.

A fair comparison holds several budgets fixed

Match training data, backbone access, answer supervision, and evaluation tasks when testing the effect of a latent module. Separately report inference FLOPs, latency, memory, and output length.

A frozen-decoder method and an end-to-end-trained system answer different engineering questions.

Treat transfer from symbolic puzzles to scientific hypothesis search as an open empirical question, with a proposed evaluation rather than an established consequence.

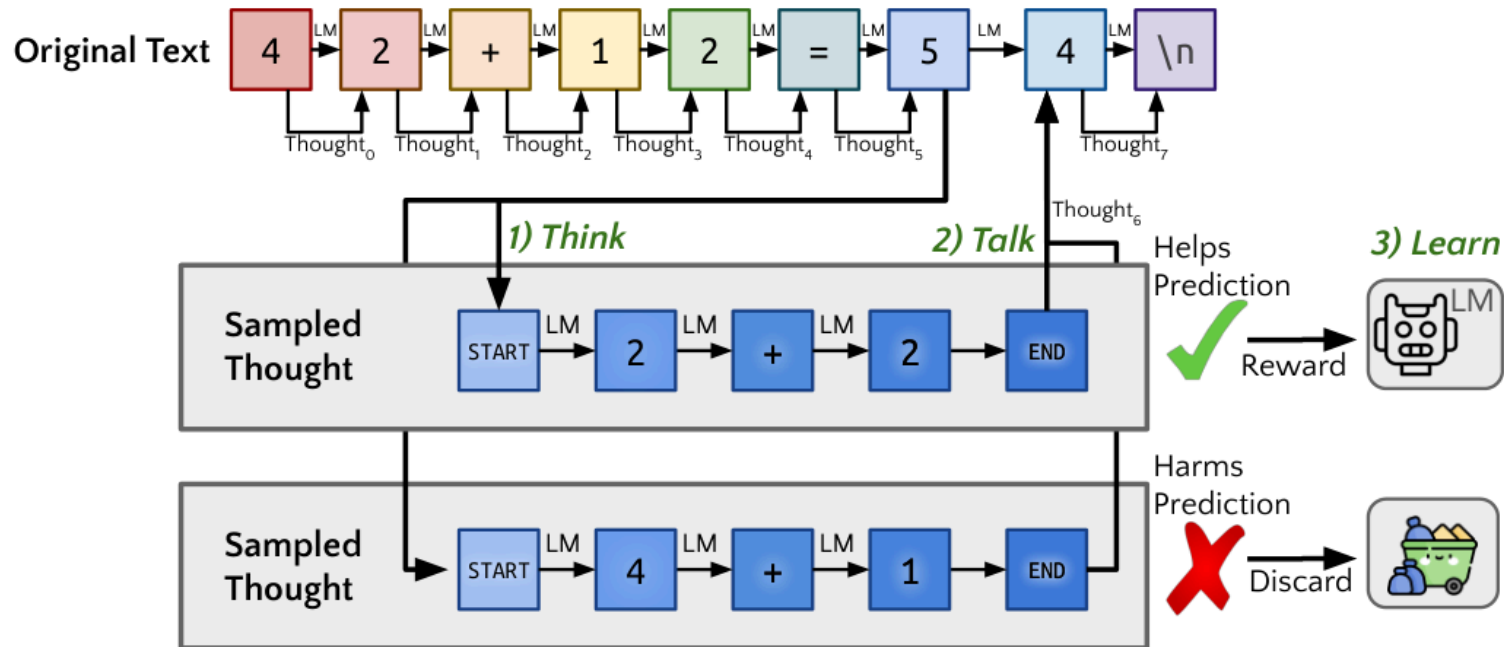
Quiet-STaR: the method

Quiet-STaR learns intermediate textual rationales that help predict subsequent ordinary text. It samples thoughts, combines predictions with and without those thoughts, and reinforces useful reasoning trajectories.

This extends reasoning supervision beyond a fixed collection of question–answer rationales.

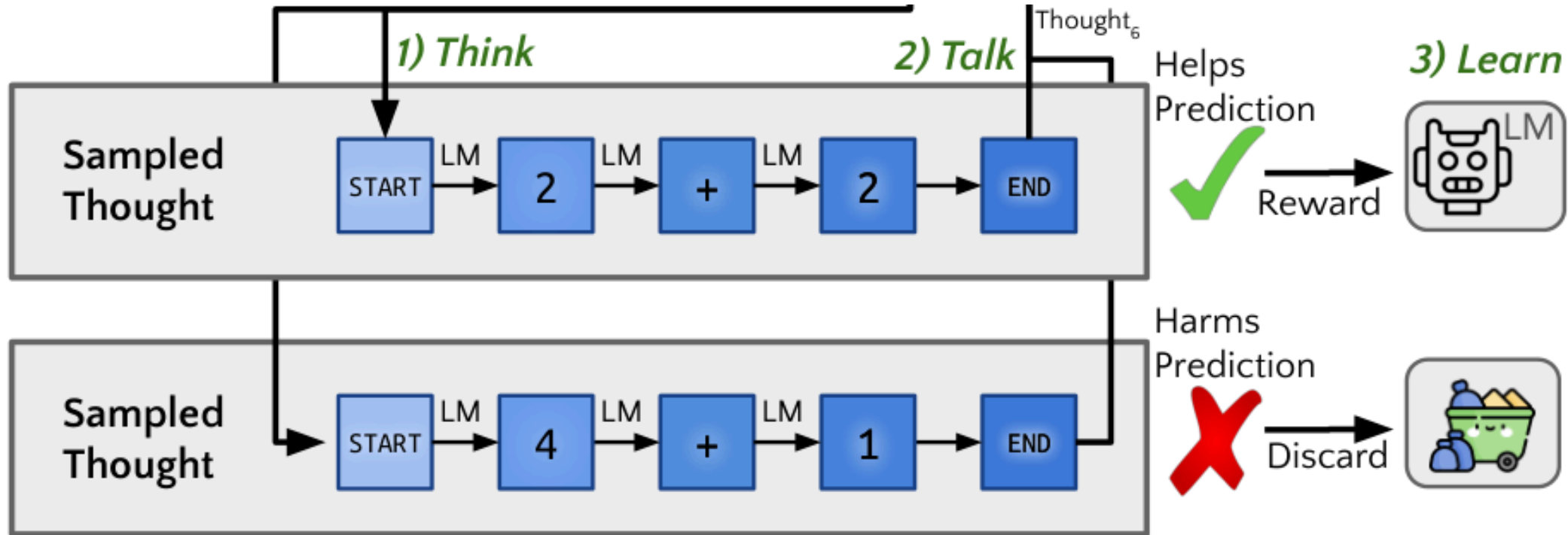
The intermediate thoughts remain token sequences in this procedure, so its contribution should be distinguished from architectures that feed continuous hidden states directly back as inputs.

Quiet-STaR: think, talk and learn



The think, talk and learn stages connect sampled rationales with future-token predictions. Reward measures predictive usefulness, which is distinct from validity of a scientific explanation.

Quiet-STaR: mechanism detail



The comparison contrasts predictions made with and without a thought. The rationale-selection signal defines an optimisation target, which is distinct from truth, causal faithfulness or experimental value.

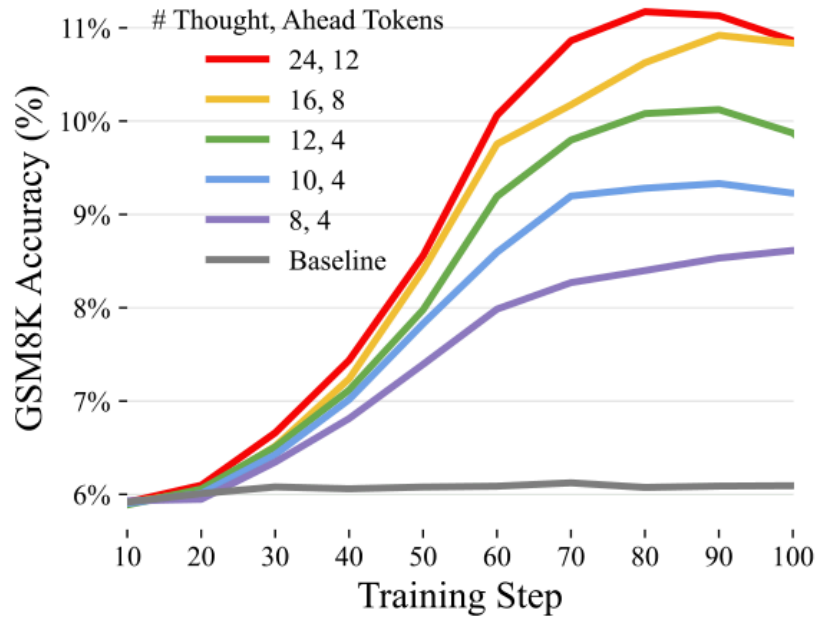
Evaluation scope

The source evaluates generalisation to reasoning benchmarks under different thought and prediction-horizon settings.

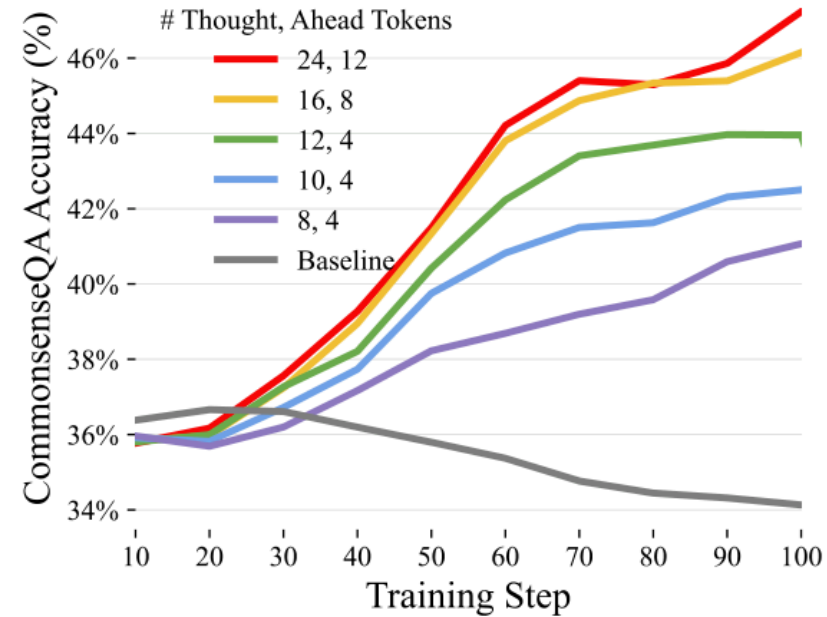
In the original curves, compare training progress and evaluation tasks while checking the legend's thought and ahead-token settings.

The result supports a particular learning procedure for useful intermediate text. It does not establish that all latent or recurrent reasoning methods share the same mechanism.

Quiet-STaR: interpretation and limits



(a) GSM8K



(b) CommonsenseQA

Predictive usefulness and explanation faithfulness are different targets. Additional thoughts consume computation, and improvements on the evaluated tasks do not automatically transfer to scientific planning or experimentation.

Representation and budget are separate choices

A method may change the intermediate representation, the number of reasoning steps, the stopping policy or the training signal. Improvements can result from any of these changes.

Compare architectures under declared training and inference budgets. Token count alone misses hidden-state computation and verifier cost.

A resource-controlled architecture comparison

Factor	Hold fixed where possible	Report separately
Data	Training and test access	Extra supervision
Model	Backbone and capacity	Trainable parameters
Inference	Allowed information	Work, latency, memory
Selection	Answer checker	Verification cost

When a factor cannot be matched, state which practical system comparison remains meaningful and which causal claim is unavailable.

A stopping rule must predict useful progress

Small changes in a hidden state can indicate convergence, but they can also indicate stagnation. A stopping signal needs validation against the desired output.

Choose a rule using development tasks, freeze it, then evaluate on held-out tasks. An oracle that sees correctness before stopping is an upper bound rather than a deployable policy.

Adaptive budgets in a synthetic task mixture

Policy	Easy 80%	Hard 20%	Mean cost / accuracy
Small budget	Cost 2; accuracy .90	Cost 2; accuracy .50	2 / .820
Large budget	Cost 8; accuracy .92	Cost 8; accuracy .85	8 / .906
Perfect routing	Use small	Use large	3.2 / .890

Perfect routing is a teaching upper bound. A real router makes errors and adds cost.

A recurrence gives a computable stopping bound

Operation	Calculation or state	Consequence
Update	$z(t+1)=0.5 z(t)+0.5; z(0)=0$	Fixed point is 1
Residual	$ 1-z(t) =2^{(-t)}$	Contraction by one half
Tolerance	$2^{(-t)} \leq 0.01$	At least seven updates
Sensitivity	$dz(t)/dz(0)=0.5^t$	At $t=7$: $1/128$

Synthetic scalar recurrence. A convergence guarantee concerns the defined state dynamics. Output correctness requires a separate relationship between the state and the task.

Illustrative recurrence: a simple state update

$$z_{t+1} = 0.5z_t + 0.5, \quad z_0 = 0$$

Consider the scalar recurrence $z_{t+1} = 0.5z_t + 0.5u$ with input signal $u = 1$ and initial state $z_0 = 0$.

This is a synthetic dynamical system, not a model of a trained language model. It isolates the effect of recurrent computation.

The recurrence approaches its fixed point when its contraction factor has magnitude below one. A coefficient above one can amplify deviations from the fixed point.

Unroll the recurrence and bound the error

$$z_t = 1 - 2^{-t}, \quad |1 - z_t| = 2^{-t}$$

The first states are 0.5, 0.75, and 0.875. The fixed point is one, and the error after t updates is 2 to the power minus t .

Additional iterations improve this particular approximation because the map is contractive. A nonlinear learned recurrence need not satisfy that condition globally.

This distinction prevents an architectural diagram from being mistaken for a guarantee that more thinking always improves the answer.

Choose an iteration budget from a tolerance

Updates t	State z_t	Absolute error	Below 0.01?
1	0.5	0.5	No
2	0.75	0.25	No
4	0.9375	0.0625	No
7	0.9921875	0.0078125	Yes

The recurrence starts at zero and approaches one. Seven updates meet the stated tolerance because the error is 2^{-t} . If each update costs one unit and decoding costs four, total cost is eleven. A small state change in a learned model still needs validation against answer quality.

Scientific transfer beyond puzzle benchmarks

A simulated system with known mechanisms and a fixed experiment menu permits a controlled comparison of explicit reasoning, continuous feedback and recurrent depth under matched training and inference budgets.

Identification accuracy, experiment efficiency, calibration and robustness to an unseen mechanism capture different consequences of a reasoning architecture. A shared hypothesis decoder helps isolate the architectural contribution.

A successful puzzle benchmark motivates this experiment; only the transfer evaluation can establish whether latent reasoning improves the scientific decision process.

Unrolled recurrence changes the gradient path

$$\frac{\partial z_t}{\partial z_0} = J_t - 1J_{t-2} \cdots J_0, \quad J_i = \frac{\partial f(z_i, x)}{\partial z_i}$$

For $z_{i+1}=f(z_i,x)$, repeated chain-rule application gives a product of state Jacobians. Shared parameters do not remove the computational depth.

Persistent contractions can erase distinctions; expansion in some directions can amplify perturbations. This is a local sensitivity analysis, not a proof about every trained recurrent model.

Probe task accuracy, perturbation sensitivity, and compute together when increasing latent iterations.

LRT separates proposal, refinement and decoding

Component	What changes?	Training role
Task-conditioned proposer	Creates base latent sequence L0	Learn the initial continuous representation
Recurrent refiner	Adds a bounded residual to L0	Learn refinements while the proposer is fixed
Frozen language decoder	Reads the refined latent sequence	Weights fixed; gradients can cross activations
At inference	Choose a bounded refinement budget	More iterations do not guarantee better answers

Latent Recurrent Thoughts (LRT), September 2026 preprint. This architecture is different from Coconut's direct hidden-state feedback.

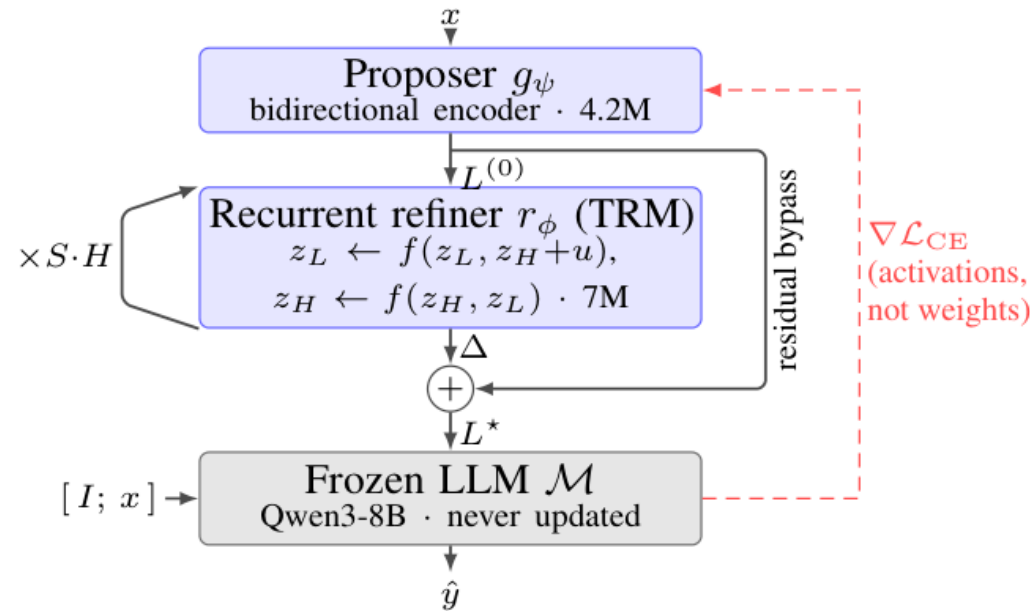
LRT: separating the proposer from refinement

Table 3 compares two components on Countdown-4 (CD4): a generic proposer scores 5.9% without refinement and 14.8% with it; a task-specific proposer scores 42.0% and 56.7%, respectively.

In Table 1, LRT reaches $56.7 \pm 1.9\%$, SoftCoT $5.9 \pm 0.2\%$, and EBM-CoT $8.4 \pm 0.1\%$ on CD4. Values are mean $\pm$ SD over three seeds with the same frozen Qwen3-8B decoder.

These results concern trained latent modules. The thinking-mode reference in Table 2 is not compute-matched. The September 2026 preprint establishes neither universal superiority nor a fixed-point guarantee.

LRT: trained modules and a frozen decoder



The proposer supplies base latents; the refiner repeatedly reads them and adds a bounded correction. Training the latent modules changes the input to the frozen decoder. This architecture does not require a visible chain of thought.

Recent methods in context

Coconut (COLM 2025). Coconut feeds a hidden state back as a continuous intermediate input. This changes the representation used during reasoning, while its training curriculum and evaluation tasks constrain claims about generality.

Latent Thinking Optimization (ICLR 2026). Uses a correctness predictor over latent states to guide latent reasoning updates. A decodable reward signal is not a faithful explanation of the underlying reasoning.

Agent Primitives (ICML 2026). Composes reusable agent operations using KV-cache communication and an organizer that chooses a configuration for each query. Attribution to collaboration or latent communication depends on matched backbones and resource budgets.

References (1/2)

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su et al. (2024). Training Large Language Models to Reason in a Continuous Latent Space

Jonas Geiping, Sean McLeish, Neel Jain et al. (2025). Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach

Eric Zelikman, Georges Harik, Yijia Shao et al. (2024). Quiet-STaR: Language Models Can Teach Themselves to Think Before Speaking

Zhaoliang Chen, Jie Fu (2026). Latent Recurrent Thoughts: Recurrent Refinement of Proposed Latents for Reasoning with Frozen LLMs

References (2/2)

[Jingcheng Hu, Yinmin Zhang, Shijie Shang et al. \(2026\). PaCoRe: Learning to Scale Test-Time Compute with Parallel Coordinated Reasoning](#)

[Hanwen Du, Yuxin Dong, Xia Ning \(2025\). Latent Thinking Optimization: Your Latent Reasoning Language Model Secretly Encodes Reward Signals in Its Latent Thoughts](#)

[Haibo Jin, Peng Kuang, Ye Yu et al. \(2026\). Agent Primitives: Reusable Latent Building Blocks for Multi-Agent Systems](#)