VARIATIONAL AUTOREASONERS: RECURSIVE REASONING BY AMORTIZING THE OTHER HALF OF EM

Anonymous authors
Paper under double-blind review

ABSTRACT

Recursive reasoning models share a set of design choices—a persistent state, a shared transformation, a prediction at every iteration, and local supervision with truncated gradients—whose computational role remains unclear. We interpret them through variational EM. Existing amortized inference amortizes only the E-step: it refines a posterior that conditions on the target, with no way to carry the result forward or to continue without the target. We introduce an instance-level M-step that adapts a per-problem memory conditioning the prior while shared weights stay fixed; the resulting inference–adaptation process inherits the monotone likelihood ascent of EM and anneals the prior toward the best explanations of the target. Variational Autoreasoners amortize this process with local variational objectives and run without the target through a recursive reasoning prior. The divergence between posterior-driven and prior-driven execution decomposes into the KL terms of the local objectives, and each design choice of recursive reasoners acquires a computational role.

1 INTRODUCTION

Recursive reasoning models (RRMs) solve structured reasoning problems such as Sudoku-Extreme (Wang et al., 2025) and ARC-AGI (Chollet, 2019) with networks far smaller than the language models they outperform (Wang et al., 2025), and have recently been extended to generation (Baek et al., 2026). Despite architectural differences, they share a set of design choices across their supervised iterations: a persistent state carried from one iteration to the next, a shared transformation and a decoded prediction at every iteration, and local supervision with gradients truncated between iterations. These choices are known to work—they yield strong accuracy and memory-efficient training—but they were arrived at empirically, and it remains unclear whether they form merely an effective recipe or express a common computational principle. In particular, it is not clear what a single iteration computes, why supervising each iteration locally suffices to train a procedure that runs for many, and how the procedure turns additional computation into better predictions.

We look for such a principle in variational EM (Dempster et al., 1977; Neal & Hinton, 1998). Reasoning toward a target y from an input x can be cast as inferring a latent explanation z of the target; EM alternates between inferring a distribution over such explanations (the E-step) and updating the model in light of them (the M-step). Amortized inference learns the E-step across problems, as in variational autoencoders (Kingma & Welling, 2014); the shortfall of its one-shot approximation is the amortization gap (Cremer et al., 2018). Semi-amortized and iterative amortized methods close this gap with computation, refining the approximation within each problem (Kim et al., 2018; Marino et al., 2018). These posterior refinement methods already combine learning across problems with iterative improvement within each—the combination that recursive reasoners exhibit.

Posterior refinement, however, leaves two central aspects of recursive reasoning unexplained. First, it improves an approximation toward a fixed target—the posterior under a model that does not change within a problem—so once the approximation has converged, additional computation has nothing left to do, and nothing the refinement has established is carried into the model that generates the next attempt. Second, the posterior being refined conditions on the target, which is useful during training but is the very answer the model must produce at test time, where only the prior remains—and in these methods the prior is consulted once. The model thus learns how to infer, yet cannot use that skill when it must actually solve a problem. An account of recursive reasoning must therefore

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.

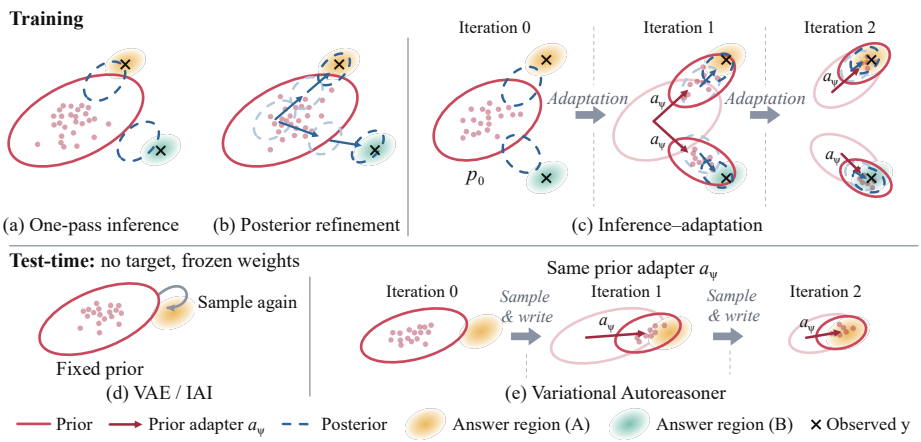


Figure 1: **(a)** One-pass inference (VAE) approximates the target-conditioned posterior. **(b)** Posterior refinement (IAI) improves inference under a fixed prior. **(c)** Inference–adaptation updates memory to carry inferred explanations into the next prior. **(d)** VAE/IAI resample an unchanged prior at test time. **(e)** The Autoreasoner amortizes (c) to run recursively from prior samples without the target.

explain both how the result of inference is carried into subsequent computation, and how a procedure learned with the target continues without it.

Our starting point is the other half of EM. The usual M-step updates parameters shared across problems, whereas recursive execution must carry information within a problem while those parameters stay fixed. We therefore introduce an *instance-level M-step*, which adapts a per-problem memory that conditions the next prior, so that each inference becomes the starting point of the next; we call the resulting alternation *inference–adaptation* (Section 2; Figure 1). Variational Autoreasoners amortize both steps—a posterior network for inference and a prior adapter for adaptation, trained with local variational objectives—and at test time run the same loop from the prior alone, forming a *recursive reasoning prior* (Section 3). We call this *process amortization*: where amortized inference learns to produce an explanation, the Autoreasoner learns how explanations are inferred and carried forward. In this framework, the design choices of RRMs acquire computational roles: the persistent state is the memory, the shared transformation is the amortized inference–adaptation step, the per-iteration prediction is the decoder reading the current explanation, and local supervision with truncated gradients is local variational training. Section 3.4 answers the questions above.

Our contributions are threefold. First, we formulate inference–adaptation, an instance-level EM process that inherits the monotone likelihood ascent of EM and, in closed form, anneals the prior toward the best explanations of the target (Section 2). Second, we introduce Variational Autoreasoners, which amortize this process with local variational objectives, run without the target as a recursive reasoning prior, and give the design choices of RRMs computational roles (Section 3). Third, on graph coloring, N-Queens, and Sudoku, Autoreasoners keep improving beyond their training depth, gain more from depth than baselines do from resampling, and keep training memory constant across horizons (Section 5).

2 Recursive Reasoning as Inference–Adaptation

Recursive reasoning lets the result of one iteration guide the computation of the next. We first formalize this with an idealized process, instance-level EM, which alternates posterior inference with memory adaptation for a single problem while shared model parameters stay fixed. The process is idealized in three respects that Section 3 later removes by amortizing the process: inference is exact, it observes the target, and memory is optimized directly.

2.1 Instance-Level EM with a Memory-Conditioned Prior

For an input x, target y, and explanation z, we condition the prior on an instance-specific memory h, while keeping shared parameters θ fixed:

$$p_\theta(y, z \mid x, h) = p_\theta(y \mid x, z)\, p_\theta(z \mid x, h).$$

Memory enters only through the prior: adaptation changes which explanations are favored without changing how they predict the target. Memory thus indexes a family of priors $\{p_\theta(z \mid x, h)\}_h$, and the process starts from an initial memory h_0 computed from the input, whose prior $p_\theta(z \mid x, h_0)$ is the model's belief before any inference on the current problem.

For a distribution $q(z)$, we use the ELBO on the target log-likelihood $\log p_\theta(y \mid x, h)$:

$$\begin{aligned}\mathcal{F}(q, h) &= \mathbb{E}_{q(z)}[\log p_\theta(y \mid x, z)] - \mathrm{KL}(q(z) \,\|\, p_\theta(z \mid x, h)) \\ &= \log p_\theta(y \mid x, h) - \mathrm{KL}(q(z) \,\|\, p_\theta(z \mid x, y, h))\,.\end{aligned}$$

The first form balances predictive fit against departure from the current prior; the second shows that the bound falls short of the target log-likelihood by exactly the error of q as a posterior. Inference–adaptation is coordinate ascent on $\mathcal{F}$ over q and h, as EM is over q and θ (Neal & Hinton, 1998).

The E-step holds memory h fixed and maximizes $\mathcal{F}$ over unrestricted q, yielding the exact posterior $q_h^\star(z) = p_\theta(z \mid x, y, h)$, at which the bound is tight; it thus combines the preferences encoded in memory with the evidence supplied by the target. The *instance-level M-step* then holds $q_h^\star$ fixed and adapts memory. Unlike a conventional parameter M-step, this update changes only the state associated with the current problem. Since the decoder does not read memory, maximizing the same objective amounts to fitting the prior to the inferred posterior:

$$h^+ \in \arg\max_{\tilde{h}} \mathcal{F}(q_h^\star, \tilde{h}) = \arg\min_{\tilde{h}} \mathrm{KL}\Big(q_h^\star(z) \,\|\, p_\theta(z \mid x, \tilde{h})\Big)\,.$$

If the prior family contains $q_h^\star$, an exact update makes this posterior the next prior. Otherwise, it returns the projection of $q_h^\star$ onto the family under this KL divergence, which is mass-covering: a family too restricted to represent several competing explanations spreads its mass over them rather than choosing among them. Iterating the two steps from h_0 produces memories $h_0, h_1, h_2, \ldots$, with h_{t+1} obtained by adapting h_t to $q_{h_t}^\star$; we call this alternation *inference–adaptation*.

2.2 The Role of Memory Adaptation

Memory adaptation is what gives additional computation something to do. With memory held fixed, the exact E-step is idempotent: repeating it returns the same posterior $p_\theta(z \mid x, y, h)$, so further iterations change nothing. Posterior refinement methods hit the same ceiling in approximate form, since they converge toward a fixed target posterior. The instance-level M-step removes this ceiling by moving the target itself—each adaptation changes the prior, and with it the next posterior.

This movement is directed. With exact inference, any memory update that improves the ELBO at the current posterior also improves the target likelihood: if $\mathcal{F}(q_h^\star, h^+) \geq \mathcal{F}(q_h^\star, h)$, then

$$\log p_\theta(y \mid x, h^+) \geq \mathcal{F}(q_h^\star, h^+) \geq \mathcal{F}(q_h^\star, h) = \log p_\theta(y \mid x, h),$$

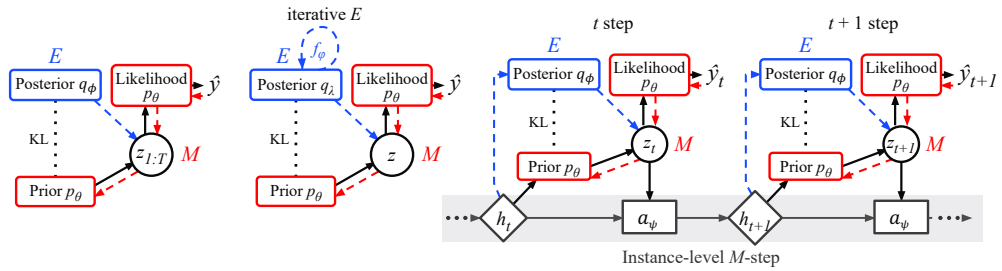
where the equality holds because the ELBO is tight at the exact posterior. This is the classical monotonicity of EM (Dempster et al., 1977; Neal & Hinton, 1998) applied to memory, and as in generalized EM, a partial adaptation suffices. The guarantee holds for the idealized process, not by itself for the learned, target-free procedure of Section 3.

When the prior family can represent the posterior, the process admits a closed form. Writing $b_t(z) = p_\theta(z \mid x, h_t)$ and $L(z) = p_\theta(y \mid x, z)$, each exact update makes the current posterior the next prior,

$$b_{t+1} \propto b_t L, \quad \text{so that} \quad b_t(z) \propto b_0(z)\, L(z)^t.$$

Since $p_\theta(y \mid x, h_t) = \mathbb{E}_{b_t}[L]$, the target likelihood rises toward $\max_z L(z)$ over the support of b_0, and the belief anneals toward the set of best explanations while preserving, within that set, the relative weights of the initial prior. Evidence thus compounds across iterations: explanations that fit the target better are progressively favored, even over those the initial prior preferred. Reusing the target overcounts it as evidence but suits the search that reasoning demands, where explanations that nearly fit are suppressed exponentially. A similar sharpening is known to aid reasoning in language models: sampling from a power of the model's distribution outperforms standard sampling (Karan & Du, 2026). Independent samples from an unchanged prior never incorporate earlier results.

The endpoint of this process is not itself the object of interest: maximizing the likelihood of a single target over memory merely concentrates the prior on that target's best explanations, a state reachable



(a) VAE / HVAE (b) IAI (c) Variational Autoreasoner

Figure 2: Inference and adaptation for conditional prediction. (a) VAE infers a latent z, while HVAE infers a hierarchy $z_{1:T}$; (b) IAI iteratively refines its posterior. (c) The Autoreasoner additionally writes explanations into memory through a_ψ, preparing the next prior and posterior. The next local ELBO trains this adaptation; at test time, the same loop uses prior samples with frozen weights. Conditioning on x and y is omitted for clarity.

only with the target in hand. What matters is the trajectory toward it. Prior adaptation provides a bridge from target-conditioned inference to target-free prediction: once the adapted prior matches the inferred posterior, it generates the same distribution of explanations without conditioning on the target again. Yet the inference that built this memory still depended on the observed target, so adaptation alone does not yield a reasoning procedure for test time. Section 3 therefore amortizes both steps, so that memory can be built from prior samples instead.

3 Variational Autoreasoners

The idealized process of Section 2 assumes exact, target-conditioned inference and explicit memory optimization in order to specify the roles of the two steps, not to prescribe an update algorithm. Variational Autoreasoners supply the algorithm by amortizing both steps with networks learned across problems: a posterior network for inference and a prior adapter for memory adaptation (Figure 2).

3.1 The Learned Inference–Adaptation Cycle

Each iteration infers an explanation under the current memory, predicts the target from it, and writes it into memory for the next iteration. Memory is initialized from the input, $h_0 = \mathrm{init}(x)$. At iteration t, the memory-conditioned prior $p_\theta(z_t \mid x, h_t)$ plays the role of the current belief of Section 2, and the posterior network produces $q_\phi(z_t \mid x, y, h_t)$, an amortized approximation of the exact posterior $q^\star_{h_t}$. We sample an explanation from it and update memory through the prior adapter a_ψ:

$$z_t \sim q_\phi(z_t \mid x, y, h_t), \qquad h_{t+1} = a_\psi(x, h_t, z_t).$$

The adapter amortizes the instance-level M-step but departs from its idealized counterpart in two respects. First, it receives a single sampled explanation rather than the full posterior, so exact distributional transfer is not assumed; instead of solving the prior-matching problem of Section 2.1, it is trained to produce memories that serve the next iteration (Section 3.2). Second, its inputs exclude the target: y can influence memory only through an explanation the model itself has drawn, which allows the same adapter to run at test time, when explanations come from the prior (Section 3.3).

3.2 Local Training

Each iteration is trained with a local ELBO that evaluates its explanation against the same target:

$$\mathcal{L}_t = \mathbb{E}_{q_t}[\log p_\theta(y \mid x, z_t)] - \mathrm{KL}(q_t \,\|\, p_\theta(\cdot \mid x, h_t)), \qquad q_t = q_\phi(\cdot \mid x, y, h_t).$$

The reconstruction term trains explanations to be predictive, and the KL term trains the prior, the only source of explanations at test time, to anticipate the target-informed posterior. Because the posterior network, prior, and decoder are shared across iterations, every latent must support a complete prediction of the target, and every memory is evaluated by another instance of the same local task. This distinguishes the unrolled procedure from hierarchical VAEs (Sønderby et al., 2016; Vahdat & Kautz, 2020; Child, 2021), whose latent layers take distinct roles and only the last is decoded.

Algorithm 1 Local training through one prior adaptation

Require: Problem x, target y, number of iterations $N \geq 2$, parameters (θ, ϕ, ψ)
$h_0 \leftarrow \text{init}(x)$
for $t = 0, \ldots, N-1$ **do**
  $q_t \leftarrow q_\phi(\cdot \mid x, y, h_t)$; $b_t \leftarrow p_\theta(\cdot \mid x, h_t)$; $z_t \sim q_t$
  $\mathcal{L}_t \leftarrow \mathbb{E}_{q_t} \log p_\theta(y \mid x, z_t) - \mathrm{KL}(q_t \,\|\, b_t)$
  Clear gradients; backpropagate $-\mathcal{L}_t$ ▷ reaches a_ψ through h_t when $t > 0$
  Update (θ, ϕ, ψ)
  if $t < N-1$ **then**
    $h_{t+1} \leftarrow a_\psi(x, \mathrm{sg}(h_t), \mathrm{sg}(z_t))$ ▷ graph kept for the next objective
  end if
end for

The local objective $\mathcal{L}_t$ evaluates inference at the incoming memory h_t; it does not evaluate the memory h_{t+1} that the adapter writes. The adapter is therefore trained through the next objective $\mathcal{L}_{t+1}$, whose gradients reach a_ψ through both the next prior and the next posterior. This gives the adapter a concrete task: prepare a context in which the next inference is more predictive and better anticipated by the prior. Preparing the next inference differs from fitting the current posterior: the idealized M-step holds the old posterior fixed and fits a prior to it, whereas the Autoreasoner recomputes the posterior under the adapted memory and scores that inference. With exact next inference, $\mathcal{L}_{t+1}$ equals the target log-likelihood $\log p_\theta(y \mid x, h_{t+1})$, the quantity the idealized M-step increases; with approximate inference, nothing guarantees that the adapted prior retains the current posterior.

As in Section 2, where each step starts from the current memory as given, we detach the adapter's inputs, $h_{t+1} = a_\psi(x, \mathrm{sg}(h_t), \mathrm{sg}(z_t))$. Information persists in memory, but its computation graph does not: $\mathcal{L}_{t+1}$ trains the current inference and the single adaptation that produced h_{t+1}, and nothing earlier. Algorithm 1 applies an optimizer update after every iteration, so retained activation storage does not grow with the number of iterations. The cost is that a memory update whose benefit appears only several iterations later receives no gradient (Williams & Peng, 1990; Tallec & Ollivier, 2017). These optimizer updates change shared weights and are distinct from the instance-level M-step, which the adapter performs on memory.

3.3 The Recursive Reasoning Prior

At test time the target is unavailable. We freeze the weights and replace posterior samples with prior samples, keeping the same adapter and decoder:

$$z_t \sim p_\theta(\cdot \mid x, h_t), \qquad h_{t+1} = a_\psi(x, h_t, z_t), \qquad \widehat{y}_t \sim p_\theta(\cdot \mid x, z_t).$$

We call this evolving latent process the **recursive reasoning prior**. Each explanation is written into memory and changes the prior from which the next one is drawn, so earlier computation shapes later computation without the target and without any weight update. Because every iteration applies the same operations, an answer can be decoded at every iteration and the loop can run beyond its training depth. Neither property guarantees that more iterations improve predictions; that depends on whether the prior-driven trajectory reproduces the posterior-driven one seen in training.

This is a genuine transfer problem: the adapter receives posterior samples during training but prior samples at test time, and the difference can compound through memory. For a frozen checkpoint, let Q_T and P_T denote the distributions of latent trajectories $z_{0:T-1}$ generated by posterior-driven and prior-driven execution from the same initial memory with the same adapter. Since memory is a deterministic function of past explanations, the chain rule of KL divergence gives (Appendix C) $\mathrm{KL}(Q_T \,\|\, P_T) = \sum_{t=0}^{T-1} \mathbb{E}_{Q_T}[\mathrm{KL}(q_\phi(\cdot \mid x, y, h_t) \,\|\, p_\theta(\cdot \mid x, h_t))]$, with each term evaluated at a memory reached by posterior sampling. We call this quantity the *process amortization gap*: it measures how far target-free execution falls from the target-conditioned process it is trained to reproduce, as the amortization gap does for a single posterior. It is the sum of the local objectives' KL terms along a training trajectory, so local training penalizes the divergence at every memory it visits, although truncation credits each adaptation only for the next term.

3.4 Design Choices of Recursive Reasoners, Revisited

Each design choice shared by RRMs (Section 1) thus acquires a computational role. The persistent state is the memory that carries the result of inference forward; the shared transformation is the amortized inference–adaptation step, identical across iterations because it amortizes a single operation; the prediction at every iteration is the decoder reading the current explanation; and local supervision with truncated gradients is local variational training, in which every iteration carries a complete objective, at the price of no credit for benefits that appear several iterations later. These roles answer the questions of Section 1. An iteration performs one inference–adaptation step. Local supervision can train a procedure that runs for many iterations because the adapter is judged by the next objective, while memory carries information across the whole trajectory. Additional computation helps to the extent that the learned loop replays, without the target, the annealing of Section 2.2, which the process amortization gap measures. The correspondence holds at the level of supervised iterations, which in existing models may span several network evaluations. For Autoreasoners it holds by construction; for existing RRMs it is an interpretation rather than a derivation.

4 Related Work

Recursive reasoning. HRM and TRM (Wang et al., 2025; Jolicoeur-Martineau, 2025) combine shared recurrent computation with deep supervision. Shared-depth computation also underlies Universal, looped, and recurrent-depth Transformers (Dehghani et al., 2018; Yang et al., 2024; Geiping et al., 2025). GRAM (Baek et al., 2026) adds stochastic latents for generation and depth/width scaling, but trains a truncated surrogate of a trajectory ELBO. The Autoreasoner instead gives each iteration a complete local ELBO and trains memory through the next objective, while truncating gradient credit across iterations.

Hierarchical latent-variable models. Ladder VAE, NVAE, and very deep VAEs organize latent groups under one joint generative objective (Sønderby et al., 2016; Vahdat & Kautz, 2020; Child, 2021). The Autoreasoner instead trains each successive latent to predict the same complete target through a shared decoder and a local ELBO (Appendix D). VRNN (Chung et al., 2015) shares transitions across distinct observations; our iterations repeatedly address one target.

Iterative variational inference. Iterative amortized inference (Marino et al., 2018) learns posterior-refinement updates, while semi-amortized VAEs (Kim et al., 2018) refine encoder outputs by differentiable optimization. Their iterative inference refines a target-conditioned posterior. We additionally adapt the next instance-specific prior through memory, learning a transition executable without the target.

5 Experiments

We abbreviate the Variational Autoreasoner as VAR in the experiments. Our goal is to evaluate inference–adaptation, rather than pursue state-of-the-art performance against HRM or TRM. With simple, moderately sized architectures, we ask whether the Autoreasoner improves structured generation and reasoning (Q1), how prior adaptation supports these gains (Q2), whether recursive depth outperforms alternatives (Q3), and which training choices enable efficient adaptation (Q4).

Tasks. To test reasoning under coupled constraints (Wang et al., 2025; Jolicoeur-Martineau, 2025; Baek et al., 2026), we use **N-Queens (NQ8/NQ10)** (Bell & Stevens, 2009), which completes a partially filled board so that no two queens share a row, column, or diagonal, and **graph coloring (GC8/GC10)** (Lemos et al., 2019), which colors the nodes of a graph so that adjacent nodes differ. NQ and GC admit multiple solutions, allowing us to evaluate solution coverage. **Sudoku generation** (Simonis, 2005) tests whether the model can produce complete 4×4 and 9×9 boards from blank inputs so that every row, column, and box contains each digit exactly once. Appendix E.1 details their construction.

Baselines. VAE (Kingma & Welling, 2014) tests one-shot inference and generation. **IAI** (Marino et al., 2018) refines the posterior during training but generates from a one-shot prior. **HVAE** (Sønderby et al., 2016; Child, 2021) uses a fixed-depth latent hierarchy with separate layer parameters. These comparisons distinguish posterior refinement and hierarchical generation from

Table 1: Structured reasoning with $N = 4$ on GC8/10 and $N = 8$ on NQ8/10. GC reports single-answer conflict sums (↓); NQ reports single-answer accuracy (%, ↑). Coverage (%, ↑) uses 20 independent draws.

	GC8		GC10		NQ8		NQ10	
Model	Conflicts	Coverage	Conflicts	Coverage	Accuracy	Coverage	Accuracy	Coverage
VAE	24.6 ± 4.5	82.3 ± 1.3	96.1 ± 11.1	39.2 ± 2.0	41.94 ± 0.45	18.35 ± 0.55	7.40 ± 1.45	1.56 ± 0.14
IAI	18.5 ± 3.0	84.0 ± 2.0	73.8 ± 47.7	41.4 ± 7.3	85.78 ± 2.12	77.91 ± 1.66	56.36 ± 1.65	41.07 ± 0.39
HVAE	28.2 ± 9.6	80.3 ± 3.0	78.4 ± 24.8	38.4 ± 3.5	98.36 ± 0.54	$\mathbf{90.96 \pm 0.47}$	73.32 ± 3.03	51.91 ± 1.17
VAR	$\mathbf{5.0 \pm 0.6}$	$\mathbf{87.7 \pm 0.4}$	$\mathbf{25.6 \pm 1.2}$	$\mathbf{48.7 \pm 0.8}$	$\mathbf{98.47 \pm 0.69}$	90.44 ± 0.09	$\mathbf{84.47 \pm 0.70}$	$\mathbf{55.82 \pm 0.14}$

the Autoreasoner's repeated inference–adaptation. All models are built from the same network blocks, and their recipes and pseudocode are in Appendices E.1 and F.

Evaluation. The training horizon N is the number of supervised iterations: recursion steps for the Autoreasoner, posterior refinements for IAI, and latent layers for HVAE. The execution depth K is the number of steps run at test time. In Section 5.1, models are trained with $N = 4$ on GC and $N = 8$ on NQ and Sudoku, and HVAE and the Autoreasoner execute at $K = N$. Each answer is a single draw. Coverage is the fraction of all valid solutions recovered by 20 independent draws per input. We report GC conflicts, NQ accuracy, and Sudoku-generation validity. Values are means and stds over seeds. OOM marks training runs that exhaust GPU memory, and baselines were given no smaller memory budgets than the Autoreasoner.

5.1 Q1. Does the Autoreasoner improve structured generation and reasoning?

With $N = 4$ on GC and $N = 8$ on NQ (Table 1), VAR has the lowest GC conflicts and highest coverage. On NQ10, it improves both accuracy and coverage over HVAE, from 73.32%/51.91% to 84.47%/55.82%. NQ8 accuracy and coverage are comparable to HVAE. The gains are thus clearer on the harder board: sampled explanations help recover more valid answers and distinct solutions.

VAR scales to longer training horizons on Sudoku (Table 2). HVAE leads on 9×9 at $N = 8$, while the Autoreasoner rises to 92.00% at $N = 16$ and 99.61% at $N = 32$; HVAE already exhausts memory at $N = 16$.

This is possible because VAR shares parameters across iterations and stops gradients between them, so each backward pass covers only one iteration and training memory stays constant as N grows. Information still persists through memory, while HVAE grows layer-specific parameters. Computation still grows with N. Section 5.4 measures the training-memory benefit, and Section 5.3 runs the trained models for more steps with fixed weights.

Table 2: Sudoku generation: Validity@1 (%). Each row for HVAE or VAR is trained separately and evaluated at $K = N$.

	Validity@1 (%) ↑	
Model	4×4	9×9
VAE	44.25 ± 3.18	0.00 ± 0.00
IAI	71.68 ± 6.25	0.00 ± 0.00
HVAE ($N = 8$)	96.70 ± 3.19	23.85 ± 1.31
HVAE ($N = 16$)	OOM	OOM
VAR ($N = 8$)	98.43 ± 0.18	13.50 ± 1.64
VAR ($N = 16$)	$\mathbf{99.95 \pm 0.01}$	92.00 ± 0.54
VAR ($N = 32$)	—	$\mathbf{99.61 \pm 0.10}$

5.2 Q2. How does prior adaptation support target-free reasoning?

We test prior adaptation, learned writes, and transfer to target-free execution.

Why adapt the prior? We build a two-mode Gaussian toy to observe prior adaptation directly. To make changes of the prior visible in latent space, we fix the decoder to $p(y \mid z) = \mathcal{N}(y; z, 0.25I)$, so latent coordinates directly represent observation means. All conditions share this data and decoder and differ only in how the prior is formed. The upper row of Figure 3 shows the idealized process of Section 2, where each exact instance-level M-step stores the posterior as the next prior. As memory is updated, samples from $p_\theta(z \mid h_t)$ move toward the two modes and the local log likelihood rises.

The lower row runs the frozen Autoreasoner without observations. Starting from prior samples, prior-only recursion moves the latents toward the two modes. Mode-region mass, the fraction of samples within either mode's 95% region, rises to 61.1% after 15 memory updates, whereas a fixed

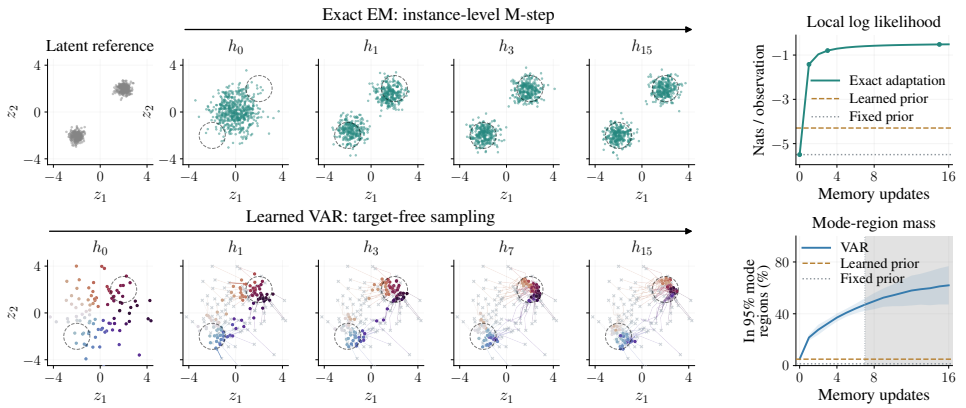


Figure 3: From exact adaptation to learned generation. Top: exact updates of the prior $p_\theta(z \mid h_t)$ from h_0 and local log likelihood. Bottom: starting from prior samples, prior-only recursion of the frozen Autoreasoner moves the latents toward the two modes (dashed circles) without observations. Crosses mark initial latents. The bottom-right panel shows mode-region mass (3-seed mean and std), and its gray shading marks execution beyond the training depth. The decoder and prior weights are fixed.

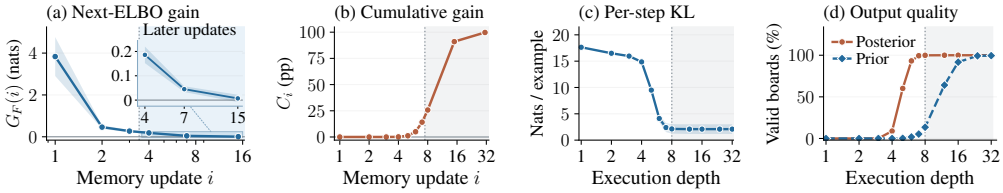


Figure 4: Learned adaptation and target-free transfer. (a) GC10 next-ELBO gain ($N = 4$), with later writes enlarged. (b) Sudoku cumulative validity gain $C_i = \sum_{j=1}^{i} U_j$ after prior writes ($N = 8$). (c,d) Sudoku per-step KL at posterior-driven memories and validity on separate posterior/prior paths. Weights are frozen, bands show std over seeds, and shading marks execution beyond training. (a,b) and (c,d) use separate evaluation runs (Appendices E.2–E.3).

$\mathcal{N}(0, I)$ prior and a single Gaussian fitted to the data reach only 1.4% and 5.0% (see Appendix E.2 for details).

Do learned memory updates help? We freeze the trained models and measure what each memory update contributes on GC10 and Sudoku. With the target available, the next-ELBO gain $G_F(i) = \mathbb{E}[F_y(h_i) - F_y(h_{i-1})]$ measures how much the i-th update improves the local ELBO $F_y(h) = \mathcal{F}(q_\phi(\cdot \mid x, y, h), h)$, with inference recomputed at the new memory and $h_i = a_\psi(x, h_{i-1}, v_i)$ written from a posterior sample. Without the target, the write utility $U_i = \mathbb{E}[V_x(a_\psi(x, h_{i-1}, v)) - V_x(h_{i-1})]$ measures how much one update raises the probability $V_x(h)$ that a fresh prior sample decodes to a valid answer, with the prefix and v drawn from the prior. Figure 4(a) shows that G_F stays positive beyond the training depth, and Figure 4(b) shows the accumulated utility on Sudoku reaching 99.71 percentage points after 31 updates. Appendix E.2 reports per-update gains.

Does the learned process transfer? To assess the *process amortization gap* (trajectory KL; Section 3.3), we compare posterior/prior execution with frozen checkpoints. Sudoku per-step KL decreases beyond training at the same posterior-driven memories (Figure 4c); GC10 shows the same trend (Appendix E.3). On Sudoku at $T = 32$, the Autoreasoner's trajectory KL is 133.2 nats, versus 581.9/582.7 for separately trained fixed-memory/constant-latent controls. Its prior-driven validity reaches 99.5%, close to 100% on the posterior path (Figure 4d). The controls fail in both modes, so they test learning the process as well as transfer.

5.3 Q3. Is recursive depth more effective than alternatives?

We test recursive depth against independent resampling, the scaling alternative for fixed-depth baselines. The Autoreasoner supports both axes and their combination. With depth K and W independent candidates per answer, we compare allocations at the nominal transition budget $B = KW$,

keeping GC10 $N = 4$ and Sudoku $N = 8$ checkpoints fixed. For $W > 1$, decoder confidence selects one whole candidate without a target or checker; Appendix E.4 also evaluates checker selection.

At $K = 32$, the Autoreasoner reduces GC10 conflicts from 25.82 to 3.82 and raises Sudoku validity from 13.50% to 99.64%. HVAE resampling at the same budget yields 66.87 conflicts and 64.43% validity. Combining depth and width on GC10 ($B = 256$) yields 2.78 conflicts, versus 3.30/4.29 for VAR depth/width alone and 63.30/35.94/64.96 for VAE/IAI/HVAE resampling. On Sudoku, HVAE reaches 97.90% at $B = 128$, while VAR's depth and hybrid policies exceed 99%. Recursive depth thus offers an effective alternative to resampling and can complement it. Further execution of the same $N = 8$ checkpoints also improves NQ10 accuracy/coverage from 84.47%/55.82% at $K = 8$ to 92.52%/60.81% at $K = 32$ (Appendix E.4, Table 9).

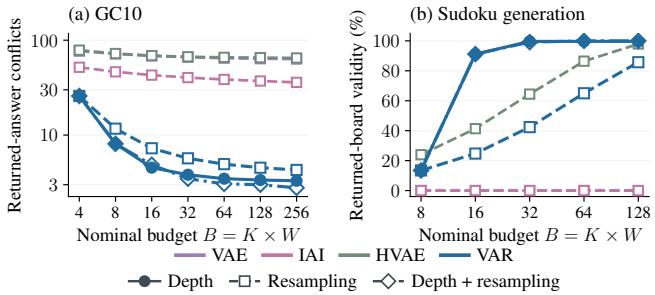


Figure 5: Depth versus independent resampling. Colors identify models; circles increase depth, squares increase width, and diamonds combine both. Decoder confidence selects candidates (Appendix E.4).

5.4 Q4. Which training choices drive performance and efficiency?

We retrain GC8 controls to test sample-dependent memory and gradient credit (Table 3). *Terminal/Repeated + BPTT* (Werbos, 1990) retain full-history gradients, with reconstruction at the final/every step. *Accumulated* retains one-write credit but updates parameters once per rollout; *Constant latent* replaces sampled writer inputs with a learned token, and *Fixed memory* bypasses the writer.

Removing writes or their sampled input lowers success to 70.7%/72.3%, supporting sample-dependent memory. The Autoreasoner achieves 97.5% at 2.88 GiB versus 94.9% at 9.43 GiB for repeated BPTT: comparable quality with much lower training memory. Training details are in Appendix E.5. With model and batch size fixed, the Autoreasoner retains $O(1)$ backpropagation activation memory in N, versus $O(N)$ for our full-BPTT implementation. Measured training allocation stays near 2.88 GiB over $N = 2$–16. This makes longer training horizons feasible under a memory limit while information continues to flow through memory; total computation still grows with depth.

Table 3: GC8 controls at $N = 4$. Success: first-answer pass rate with canonical coloring. Memory: peak allocated GPU memory during training at a fixed batch.

Condition	Success (%) ↑	Memory (GiB) ↓
Terminal + BPTT	75.0 ± 5.8	8.95
Repeated + BPTT	94.9 ± 1.2	9.43
Accumulated	96.5 ± 2.0	3.03
Constant latent	72.3 ± 1.5	2.88
Fixed memory	70.7 ± 0.9	1.46
Autoreasoner	97.5 ± 1.3	2.88

6 Conclusion & Limitations

We introduced inference–adaptation: an instance-level EM formulation in which inferred explanations reshape the prior for subsequent inference. The Autoreasoner amortizes this process with a memory adapter and local variational objectives, enabling target-free recursive generation. A trajectory-KL decomposition connects local posterior–prior matching to the training–generation gap. Experiments support sample-dependent memory and show that recursive depth can complement independent sampling, while short gradient histories make longer horizons feasible under memory constraints. Our evaluation focuses on moderate-scale structured tasks; transfer to larger, less constrained domains remains to be tested. Gains can also saturate with depth, motivating adaptive stopping and instance-dependent compute allocation.

AI USE STATEMENT

We used generative AI tools only to polish the writing, such as correcting grammar and typos and improving the flow of sentences. We did not use them for any other part of this work. All authors reviewed the edited text and take responsibility for the final content.

ETHICS STATEMENT

This work aims to understand and improve recursive reasoning in machine learning models by interpreting their computation through variational EM. Our experiments use only synthetic puzzles (N-Queens, graph coloring, and Sudoku) and a two-dimensional Gaussian toy, with no human subjects or personal data. To the best of our knowledge, this work raises no specific ethical concerns.

REFERENCES

Junyeob Baek, Mingyu Jo, Minsu Kim, Mengye Ren, Yoshua Bengio, and Sungjin Ahn. Generative recursive reasoning. *arXiv preprint arXiv:2605.19376*, 2026. URL https://arxiv.org/abs/2605.19376.

Jordan Bell and Brett Stevens. A survey of known results and research areas for n-queens. *Discrete mathematics*, 309(1):1–31, 2009.

Rewon Child. Very deep VAEs generalize autoregressive models and can outperform them on images. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=RLRXCV6DbEJ.

François Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/hash/b618c3210e934362ac261db280128c22-Abstract.html.

Chris Cremer, Xuechen Li, and David Duvenaud. Inference suboptimality in variational autoencoders. In *International conference on machine learning*, pp. 1078–1086. PMLR, 2018.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977. URL https://doi.org/10.1111/j.2517-6161.1977.tb01600.x.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *Advances in Neural Information Processing Systems*, 38: 41340–41391, 2025.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv preprint arXiv:2510.04871*, 2025. URL https://arxiv.org/abs/2510.04871.

Aayush Karan and Yilun Du. Reasoning with sampling: Your base model is smarter than you think. In *International Conference on Learning Representations*, volume 2026, pp. 476–494, 2026.

Yoon Kim, Sam Wiseman, Andrew Miller, David Sontag, and Alexander Rush. Semi-amortized variational autoencoders. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2678–2687. PMLR, 2018. URL https://proceedings.mlr.press/v80/kim18e.html.

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations*, 2014. URL https://arxiv.org/abs/1312.6114.

Henrique Lemos, Marcelo Prates, Pedro Avelar, and Luis Lamb. Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 879–885. IEEE, 2019.

Joe Marino, Yisong Yue, and Stephan Mandt. Iterative amortized inference. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3403–3412. PMLR, 2018. URL https://proceedings.mlr.press/v80/marino18a.html.

Radford M. Neal and Geoffrey E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, pp. 355–368. Springer, 1998. URL https://doi.org/10.1007/978-94-011-5014-9_12.

Helmut Simonis. Sudoku as a constraint problem. In *CP Workshop on modeling and reformulating Constraint Satisfaction Problems*, volume 12, pp. 13–27. Citeseer Sitges, Spain, 2005.

Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/hash/6ae07dcb33ec3b7c814df797cbda0f87-Abstract.html.

Corentin Tallec and Yann Ollivier. Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209*, 2017.

Arash Vahdat and Jan Kautz. NVAE: A deep hierarchical variational autoencoder. In *Advances in Neural Information Processing Systems*, volume 33, pp. 19667–19679. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/e3b21256183cf7c2c7a66be163579d37-Abstract.html.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025. URL https://arxiv.org/abs/2506.21734.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Ronald J Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 2(4):490–501, 1990.

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International conference on learning representations*, volume 2024, pp. 42195–42214, 2024.

A Exact Inference–Adaptation

Likelihood ascent. Fix (x, y, θ) and write $b_h(z) = p_\theta(z \mid x, h)$, $L(z) = p_\theta(y \mid x, z)$, and $Z(h) = \int b_h(z)L(z)\,dz$. Assume $0 < Z(h) < \infty$ and well-defined expectations and divergences. The exact posterior is $q_h^\star = b_h L/Z(h)$, and the variational identity gives

$$\mathcal{F}(q, h) = \log Z(h) - \mathrm{KL}(q\|q_h^\star).$$

Because the decoder is independent of memory, increasing $\mathcal{F}(q, h)$ at fixed q is equivalent to reducing $\mathrm{KL}(q\|b_h)$. For $q = q_h^\star$, any such update satisfies

$$\log Z(h^+) \geq \mathcal{F}(q_h^\star, h^+) \geq \mathcal{F}(q_h^\star, h) = \log Z(h).$$

This proves the claim in Section 2.2, including generalized M-steps and restricted prior families. Exact inference makes the old bound tight; approximate inference does not provide this guarantee. The result concerns adaptation to a known target with shared weights fixed, not target-free execution.

Repeated adaptation. If the memory family can represent each exact posterior, copying it into the next prior gives $b_{t+1} = b_t L/\mathbb{E}_{b_t} L$. Induction yields

$$b_t(z) = \frac{b_0(z)L(z)^t}{\int b_0(u)L(u)^t\,du}.$$

For a finite latent space, let $S = \{z : b_0(z) > 0\}$ and $A = \arg\max_{z\in S} L(z)$, with positive maximum. Dividing numerator and denominator by $\max_{z\in S} L(z)^t$ gives

$$\lim_{t\to\infty} b_t(z) = \frac{b_0(z)\mathbf{1}\{z \in A\}}{\sum_{u\in A} b_0(u)}, \qquad \lim_{t\to\infty} Z(h_t) = \max_{z\in S} L(z).$$

The limit preserves initial relative weights among maximizers but cannot expand the initial support. It reuses the same target as an optimization signal; it is not Bayesian updating on newly observed independent evidence. Continuous latent spaces require their corresponding support and integrability conditions; the point-mass formula above is a finite-space statement.

B Learning the Prior Adapter

For detached incoming memory and sample, let $h' = a_\psi(x, \mathrm{sg}(h), \mathrm{sg}(v))$, $q' = q_\phi(\cdot \mid x, y, h')$, and $b' = p_\theta(\cdot \mid x, h')$. The next local objective is

$$\mathcal{L}_{\mathrm{next}}(h') = \mathbb{E}_{q'} \log p_\theta(y \mid x, z) - \mathrm{KL}(q'\|b').$$

Gradients reach the adapter through both q' and b'. Thus the objective evaluates newly recomputed inference, rather than fitting a cached posterior. This is the predictive adaptation in Section 3.2.

In Algorithm 1, backward consumes the preceding adapter graph before the optimizer changes weights. The next write then uses the updated adapter with the detached memory and latent sample from the current iteration. Its output graph is retained for the next objective. There is no preceding write at $t = 0$, and N reads supervise $N - 1$ writes. For fixed modules, state size, and batch, only one write's gradient history is retained, although forward computation and optimizer work grow with N. The accumulated-gradient and full-BPTT alternatives are defined in Appendix E.5.

C Transfer from Posterior to Prior Trajectories

Fix (x, y) and a frozen checkpoint with a deterministic initializer and adapter. For any latent prefix, both execution modes compute the same memory. Their trajectory densities are

$$Q_T(z_{0:T-1}) = \prod_{t=0}^{T-1} q_\phi(z_t \mid x, y, h_t), \qquad P_T(z_{0:T-1}) = \prod_{t=0}^{T-1} p_\theta(z_t \mid x, h_t).$$

Assuming $Q_T \ll P_T$, expanding the log density ratio and conditioning on each prefix yields the chain rule used in Section 3.3:

$$\mathrm{KL}(Q_T\|P_T) = \mathbb{E}_{Q_T} \sum_{t=0}^{T-1} \log \frac{q_\phi(z_t \mid x, y, h_t)}{p_\theta(z_t \mid x, h_t)} = \sum_{t=0}^{T-1} \mathbb{E}_{Q_T} \mathrm{KL}(q_t\|b_t).$$

Each conditional KL is evaluated at a posterior-driven memory. The cumulative quantity includes every step, even when only selected steps are plotted.

D Relation to Hierarchical VAEs

Unrolling the Autoreasoner for T reads defines a hierarchical generator,

$$p(y, z_{0:T-1} \mid x) = p_\theta(y \mid x, z_{T-1}) \prod_{t=0}^{T-1} p_\theta(z_t \mid x, h_t), \qquad h_{t+1} = a_\psi(x, h_t, z_t).$$

The distinction in Section 4 therefore concerns learning, not whether the unrolled graph is hierarchical. With Q_T as above, a terminal hierarchical ELBO and the expected sum of local objectives are, respectively,

$$\mathcal{E}_T = \mathbb{E}_{Q_T} \log p_\theta(y \mid x, z_{T-1}) - \sum_{t=0}^{T-1} \mathbb{E}_{Q_T} \mathrm{KL}(q_t \| b_t),$$

$$\mathcal{J}_T = \sum_{t=0}^{T-1} \mathbb{E}_{Q_T} \left[\log p_\theta(y \mid x, z_t) - \mathrm{KL}(q_t \| b_t) \right].$$

The terminal objective evaluates one final prediction; the local objectives require every latent to predict the same target and train each write through the next inference. The latter is not the terminal hierarchical ELBO. Online training additionally changes shared weights between local objectives.

Ladder VAE already combines data-dependent information with generative priors to infer hierarchical latents (Sønderby et al., 2016); prior-informed inference itself is not the distinction. Traversing latent layers also differs from repeatedly refining a posterior under a fixed generative model (Marino et al., 2018). The Autoreasoner changes the prior context between local inference problems. Parameter sharing, repeated supervision, and gradient truncation are separate choices: Section 5.1 compares the reported HVAE baseline, while Section 5.4 holds the Autoreasoner backbone fixed to compare supervision and gradient history. Appendix F summarizes the respective computations.

E Experimental Details

E.1 Tasks, evaluation, and training recipes

Tasks. Following Baek et al. (2026), NQ8 inputs remove 5–7 queens from a complete 8×8 solution, and NQ10 inputs remove 7–9 from a 10×10 solution. Accuracy requires a valid completion of all constraints. GC8/GC10 use random graphs with chromatic number three (Lemos et al., 2019), 8/10 nodes, and 255/192 test graphs. Reference colorings use canonical color order: colors are relabeled by first appearance, so color permutations are not distinct solutions. Sudoku generation receives a blank board. For 9×9, training uses 50,000 complete solutions from the Sudoku-Extreme training split (Wang et al., 2025; Baek et al., 2026); no puzzle clues are provided. For 4×4, all 288 valid boards are enumerated and split into 256 training and 32 test solutions. Validity checks every row, column, box, and digit range; it is not exact match to a held-out board.

Single-answer quality and coverage. Every draw in Section 5.1 is scored independently, without candidate selection. GC conflict sums are averaged over 20 repeated draws per test graph within each training seed; this estimates the $W = 1$ policy, not best-of-20. Sudoku reports the valid fraction of individually generated boards, with 10,000-board evaluations unless a larger bank is specified below. GC uses five training seeds, and NQ and Sudoku use three. Reported uncertainty is sample std over seeds.

Coverage instead uses the set of 20 generated answers per input. For GC, with canonical reference solutions $S(x)$ and distinct generated tuples $\widehat{S}_{20}(x)$, it is

$$100 \, \frac{\sum_{x \in \mathcal{D}_{\text{test}}} |\widehat{S}_{20}(x) \cap S(x)|}{\sum_{x \in \mathcal{D}_{\text{test}}} |S(x)|}.$$

Duplicates count once, noncanonical outputs are not repaired, and graph automorphisms are not quotiented out. This is solution-count-weighted coverage, not the fraction of inputs with at least one successful draw. NQ coverage likewise uses 20 independent completions per input; its scores are reported separately from single-answer accuracy.

Training and execution horizons. Primary horizons are $N = 4$ for GC and $N = 8$ for NQ/Sudoku. HVAE and the Autoreasoner execute at $K = N$ in Section 5.1; VAE and IAI generate in one step, although IAI refines its training posterior N times. Larger-N GC and Sudoku-generation rows use separately trained models. NQ depth extensions instead reuse the same $N = 8$ checkpoints. At $N = K = 8$, VAR gives 4.0 ± 1.3 conflicts and $86.7 \pm 1.1\%$ coverage on GC8 (three seeds), and 7.5 ± 5.2 conflicts and $52.5 \pm 1.3\%$ coverage on GC10 (five seeds).

Recipes and comparison scope. Baseline recipes use learning rate 10^{-4}, weight decay 0.1, and dropout 0. Their KL coefficients are $1/81$ on GC; $0.2/81$ for VAE/IAI and 0.0078125 for HVAE on Sudoku 9×9; and 0.0125/0.0396, respectively, on 4×4. On NQ8/NQ10, the KL coefficients are 0.04/0.02 for VAE, 0.012 for IAI, and 0.015625/0.004 for HVAE. VAE uses learning rates 6×10^{-4}/3×10^{-4} and IAI 1.5×10^{-4}, with free bits 1.0/off and 0.3/1.0, respectively. GC KL warm-up lasts 18,000 steps; 4×4 warm-up lasts 7,800. HVAE has distinct parameters at each latent layer and applies stage-wise KL balancing during KL warm-up. On NQ10, it also balances the posterior and prior KL gradients with $\alpha = 0.7$. The GC IAI recipe uses direct posterior updates with layer-normalized inputs. All five seeds, including the high-conflict GC10 seed, are retained.

The Autoreasoner uses GC learning rate 5×10^{-4}, dropout 0.1, reconstruction scaling $\beta = 8$, free bits 0.3, and KL coefficient $1/81$; GC10 uses weight decay 1.0. Sudoku uses learning rate 10^{-4}, dropout 0, and $\beta = 4$, with free bits 18.2 and KL coefficient $0.5/81$ for 9×9, or 7.2 and 0.0156 for 4×4. NQ uses learning rate 10^{-4}, dropout 0, $\beta = 2$, free bits 1.92/3.0, and KL coefficient 0.015625/0.01 for NQ8/NQ10. Reconstruction scaling and free bits change the effective objective beyond the displayed KL coefficient. Evaluation uses final-checkpoint EMA, with decoder mixing disabled. Structured-task initial memory is a fixed buffer; x conditions subsequent reads.

The 4×4 backbone has hidden width 512 and latent width 128; the Autoreasoner has eight memory layers and the baselines six. Training lasts 50 epochs. On 9×9, the reported HVAE uses six memory layers and latent width 192, versus the Autoreasoner's eight layers and width 128. On NQ, all models use hidden width 512 and a 64-dimensional latent per token. Modules have two layers, except the four-layer VAE encoder and the four-layer VAE and IAI decoders. On NQ10, VAE and IAI receive twice as many updates as HVAE and the Autoreasoner. On NQ, all models use AdamAtan2 with $\beta = (0.9, 0.95)$, global batch 128, 2,000 LR warm-up steps, and EMA decay 0.999. NQ8 trains for 2,240 epochs, while NQ10 trains VAE and IAI for 200 epochs and HVAE and the Autoreasoner for 100. KL warm-up lasts 10K/40K steps for VAE and 31.2K for HVAE. NQ8 IAI uses a cosine schedule, and NQ10 VAE clips gradients at 1.0. GC IAI selection used early-seed results. These comparisons are conditional on the reported recipes, not an independently validation-selected comparison under equal tuning.

OOM entries. IAI and HVAE exhaust memory at $N = 8$ on both GC tasks; HVAE also exhausts memory at $N = 16, 32$ on both Sudoku sizes. Baselines received GPU-memory budgets no smaller than the Autoreasoner's. These entries describe the attempted configurations, not universal architectural limits. The controlled GPU-memory profiles in Appendix E.5 use their own explicitly fixed hardware and batch.

E.2 Exact and learned prior adaptation

Gaussian toy and exact update. For Figure 3, draw z from an equal mixture of $\mathcal{N}((-2,-2), 0.09I)$ and $\mathcal{N}((2,2), 0.09I)$, then $y \mid z \sim \mathcal{N}(z, 0.25I)$. There are 4,096 training and 512 held-out observations; x is absent. Every condition uses the same fixed decoder. Memory $h = (m, \log v)$ specifies a diagonal Gaussian prior. Starting from $m_0 = 0, v_0 = 1$, exact posterior-to-prior updates are

$$v_{t+1,j} = (v_{t,j}^{-1} + 4)^{-1}, \qquad m_{t+1,j} = v_{t+1,j}(m_{t,j}/v_{t,j} + 4y_j).$$

These solve the memory M-step in closed form. The upper snapshots pool one sample per observation-specific prior; no individual Gaussian becomes bimodal. Local log likelihood is evaluated analytically as $\log \mathcal{N}(y; m, \mathrm{diag}(v) + 0.25I)$ and averaged over observations. It is target-conditioned because the scored observation also determines its adapted memory.

The fixed control retains $\mathcal{N}(0, I)$. The learned shared prior uses the training-data marginal MLE: mean $\widehat{m}_j = \mathrm{mean}_n\, y_{n,j}$ and variance $\widehat{v}_j = \mathrm{var}_n\, y_{n,j} - 0.25$, using divisor 4,096 for the empirical

variance. Both fitted variances are positive. Its mean is approximately $(-0.0364, -0.0165)$ and variance $(4.0015, 4.0922)$; it is frozen during evaluation. The exact and fixed conditions share their initial prior. The neural Autoreasoner starts from its own learned initializer instead.

Learned sampler and scoring. The toy Autoreasoner has a two-dimensional latent, eight-dimensional memory, trainable h_0, and width-32 tanh prior, posterior, and writer networks. Gaussian variances use softplus plus 0.001. Three seeds (42/43/44) train for 2,000 minibatches of 128 observations with Adam at 10^{-3}, eight local ELBO updates per minibatch, and the one-write gradient boundary. Reconstruction expectations and KLs are analytic, without rescaling or free bits. The decoder stays fixed. Non-EMA checkpoints are frozen for prior-only execution, which never receives an observed target.

Each conditional read is Gaussian, but random histories create a mixture over memories. Mode-region mass is the probability of a latent falling within either true component's 95% disk (radius 0.7343); the component parameters are used only for evaluation. It measures access to the modes, not balanced mode coverage or full density recovery. We evaluate 4,096 trajectories per seed. Mass grows from $5.08 \pm 0.20\%$ to $47.03 \pm 4.84\%$ after seven writes and $61.13 \pm 13.94\%$ after 15. Holding the same checkpoint's memory at h_0, with paired latent noise, gives $4.83 \pm 0.24\%$ after 15 writes. This paired intervention isolates execution of the writer from its learned initialization. Fixed/shared-learned Gaussian masses are 1.3972%/4.9557%, computed by numerical quadrature. The comparison is limited to these Gaussian prior controls.

The lower snapshots show the first 96 seed-42 trajectories, without outcome-based selection. Gray reference points are held-out ground-truth latents; display circles enclose 99.9% component mass for visibility. Scoring uses the smaller 95% disks and retains all samples, including those outside the displayed axes.

Next-ELBO gain and write utility. Figure 4(a,b) uses the frozen checkpoints of Section 5.1: GC10 $N = 4$ (five seeds, all 192 test graphs) and Sudoku $N = 8$ (three seeds, a fixed 256-board subset). Each example has two prefix trajectories and four write probes per prefix. G_F uses posterior prefixes and posterior writes; U_i uses prior prefixes and prior writes. They evaluate different state distributions.

At each memory, the posterior is recomputed. The standard ELBO uses an analytic diagonal-Gaussian KL and 16 fresh posterior samples for reconstruction, excluding training rescaling and free bits. Utility estimates each branch's single-answer validity using 32 fresh prior samples and argmax decoding. Evaluation samples are independent of the write probe; paired branches use common random numbers. Measurements are averaged within each seed before seed means and stds are computed. GC validity includes canonical color order. Write i maps h_{i-1} to h_i; training supervises writes $1, \ldots, N-1$.

Table 4: Local write diagnostics (mean ± std): GC10 $N = 4$, five seeds; Sudoku $N = 8$, three seeds. G_F uses posterior writes; U_i uses prior writes (pp: percentage points). $*$ marks writes beyond training.

Task	Write i	G_F (nats)	U_i (pp)
GC10	1	3.835 ± 0.931	45.56 ± 9.78
	2	0.462 ± 0.084	7.46 ± 1.54
	3	0.271 ± 0.061	4.50 ± 0.62
	4^*	0.186 ± 0.034	2.11 ± 0.71
	7^*	0.045 ± 0.012	0.54 ± 0.29
	15^*	0.007 ± 0.015	0.11 ± 0.09
Sudoku	1	25.191 ± 0.469	0.00 ± 0.00
	2	33.217 ± 0.732	0.00 ± 0.00
	3	49.078 ± 0.527	0.00 ± 0.00
	4	45.916 ± 2.097	0.12 ± 0.10
	5	17.097 ± 0.633	1.08 ± 0.26
	6	3.085 ± 0.387	3.76 ± 0.42
	7	0.367 ± 0.027	8.45 ± 1.04
	8^*	0.057 ± 0.025	11.76 ± 0.17
	15^*	-0.008 ± 0.009	3.59 ± 0.69
	31^*	0.002 ± 0.003	-0.06 ± 0.12

We estimate cumulative utility in Figure 4(b) as the difference between mean validity after i prior-driven updates and mean validity at the initial memory. Only means are shown. Positive C_i demonstrates improvement over initialization; it does not mean every late write helps. In particular, late Sudoku gains approach zero and can be negative (Table 4). Likewise, positive G_F with approximate inference does not certify likelihood ascent.

E.3 Posterior–prior transfer diagnostics

The trajectory study in Figure 4(c,d) freezes the primary checkpoints and executes both sampling modes to $T = 4N$: GC10 uses 192 graphs and five seeds, Sudoku 256 boards and three seeds. This evaluation uses separate draws from the local-gain study above and the candidate banks of Section 5.3. At each posterior-driven prefix, we compute the analytic conditional Gaussian KL; summing measurements at every step gives trajectory KL in nats per example. Read depth is $K = t + 1$.

Table 5: Transfer at initialization, the training horizon, and $4N$ reads. KL is in nats (mean ± std over seeds); validity columns report means. Trajectory KL accumulates every measured step.

Task	Reads T	Per-step KL	Trajectory KL	Posterior valid (%)	Prior valid (%)
GC10	1	2.42 ± 0.71	2.42 ± 0.71	46.0	31.9
	4	0.21 ± 0.01	3.18 ± 0.71	96.7	89.2
	16	0.12 ± 0.01	4.86 ± 0.63	99.8	95.9
Sudoku	1	17.64 ± 0.05	17.64 ± 0.05	0.0	0.0
	8	2.13 ± 0.90	83.10 ± 2.60	99.9	13.5
	32	2.09 ± 0.93	133.20 ± 24.80	100.0	99.5

The local KL terms decrease on both tasks, while their cumulative sum grows (Table 5). On Sudoku at $T = 32$, the prior reaches 99.5% validity versus 100% on the posterior path. These are output-quality comparisons, not evidence of identical latent paths.

Separately trained Sudoku fixed-memory and constant-latent controls use the primary $N = 8$ backbone and recipe (three seeds; definitions in Appendix E.5). Their conditional KL stays near 18.2 nats; trajectory KL is 145.5/145.7 at $T = 8$ and 581.9/582.7 at $T = 32$, respectively. Both have zero observed target-exact posterior decodes and zero prior validity. They therefore test learning the prediction process as well as transfer, rather than isolating transfer failure alone.

E.4 DEPTH VERSUS INDEPENDENT RESAMPLING

Frozen models and allocation. Figure 5 fixes GC10 training at $N = 4$ and Sudoku at $N = 8$. At nominal budget $B = KW$, Autoreasoner depth uses $(K, W) = (B, 1)$ and width uses $(N, B/N)$. HVAE keeps its full trained hierarchy; VAE/IAI generate each candidate in one step. GC hybrid allocations for $B = 4, 8, 16, 32, 64, 128, 256$ are $(4, 1), (8, 1), (8, 2), (16, 2), (16, 4), (32, 4), (32, 8)$. Sudoku allocations for $B = 8, 16, 32, 64, 128$ are $(8, 1), (16, 1), (16, 2), (32, 2), (32, 4)$. These schedules are fixed before policy evaluation. Equal KW does not equate FLOPs, latency, or model size.

Whole-candidate selection and estimation. For $W = 1$, the sole candidate is returned. For $W > 1$, decoder confidence sums the log probabilities of the decoded entries and selects the highest-scoring whole answer, without a target or checker. The secondary policy selects minimum GC conflicts or the first valid Sudoku board. Every candidate is generated before selection; neither early stopping nor componentwise voting is used.

We evaluate uniform subsets of W distinct candidates from a finite iid bank of size M. Rank candidates by the selection score with a fixed tie order. If $g_{(j)}$ is the task score at rank j, the exact conditional bank average is

$$\widehat{g}(W) = \sum_{j=1}^{M-W+1} g_{(j)} \frac{\binom{M-j}{W-1}}{\binom{M}{W}}.$$

For checker-selected Sudoku validity, this reduces to $1 - \binom{M-s}{W}/\binom{M}{W}$ when s bank members are valid. These are finite-bank estimates, not exact population probabilities. We average within each training seed, then report seed mean and std.

GC10 uses all 192 test graphs and five seeds: banks contain 128 candidates per graph at $K = 4$, 512 for one-step VAE/IAI, and 20 at deeper recursive depths. Sudoku uses three seeds, 200,000 boards per seed/depth for depth and checker policies, and separate 40,000-board confidence banks. Different banks explain small differences in $W = 1$ anchors. VAE has no valid boards in these Sudoku banks; IAI has one in one seed's 200,000-board bank and none in the confidence banks. Rounded zeros do not assert zero population success.

Table 6: GC10 without checker-based selection: conflict sum. Mean ± std over seeds; all seeds are included. Depth has no candidate selection.

Policy	$B = 4$	$B = 8$	$B = 16$	$B = 32$	$B = 64$	$B = 128$	$B = 256$
VAE width	76.67 ± 10.91	71.65 ± 11.21	68.32 ± 11.41	66.12 ± 11.45	64.64 ± 11.35	63.78 ± 11.24	63.30 ± 11.35
IAI width	51.83 ± 50.60	46.42 ± 50.65	42.85 ± 50.41	40.37 ± 50.12	38.57 ± 49.85	37.14 ± 49.58	35.94 ± 49.58
HVAE width	78.51 ± 24.58	72.35 ± 25.24	68.81 ± 25.70	66.87 ± 26.17	65.95 ± 26.78	65.45 ± 27.24	64.96 ± 27.25
Autoreasoner depth	25.82 ± 1.24	8.09 ± 1.25	4.54 ± 1.28	3.82 ± 1.05	3.47 ± 1.16	3.36 ± 0.89	3.30 ± 1.05
Autoreasoner width	25.82 ± 1.24	11.69 ± 0.92	7.26 ± 1.25	5.69 ± 1.18	4.94 ± 1.02	4.53 ± 0.95	4.29 ± 1.11
Autoreasoner hybrid	25.82 ± 1.24	8.09 ± 1.25	4.87 ± 1.32	3.47 ± 1.02	3.06 ± 1.07	3.00 ± 1.00	2.78 ± 1.16

Table 7: Sudoku without checker-based selection: returned-board validity (%). Mean ± std over seeds; all seeds are included. Depth has no candidate selection.

Policy	$B = 8$	$B = 16$	$B = 32$	$B = 64$	$B = 128$
VAE width	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
IAI width	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
HVAE width	23.960 ± 1.430	41.360 ± 2.070	64.430 ± 2.370	86.430 ± 1.730	97.900 ± 0.520
Autoreasoner depth	13.500 ± 1.640	91.100 ± 1.660	99.640 ± 0.280	99.670 ± 0.260	99.680 ± 0.250
Autoreasoner width	13.440 ± 1.680	24.720 ± 2.860	42.380 ± 4.220	65.000 ± 4.840	85.740 ± 3.650
Autoreasoner hybrid	13.440 ± 1.680	91.150 ± 1.750	99.140 ± 0.350	100.000 ± 0.000	100.000 ± 0.000

Effect of checker selection. Table 8 reports every policy at the largest tested budget under both selectors. A checker can substantially improve resampling, especially on GC10; thus confidence-selected depth gains are not a claim that depth always dominates width under an ideal selector. Single-trajectory scores do not depend on selection.

Table 8: Selection at the largest tested nominal budget (mean ± std). GC10: conflicts (↓); Sudoku: validity (%, ↑). Depth uses $W = 1$ under either selector.

	GC10, $B = 256$		Sudoku, $B = 128$	
Policy	Confidence	Checker	Confidence	Checker
VAE width	63.30 ± 11.35	5.69 ± 1.89	0.000 ± 0.000	0.000 ± 0.000
IAI width	35.94 ± 49.58	6.67 ± 11.20	0.000 ± 0.000	0.020 ± 0.040
HVAE width	64.96 ± 27.25	15.06 ± 7.85	97.900 ± 0.520	98.690 ± 0.360
Autoreasoner depth	3.30 ± 1.05	3.30 ± 1.05	99.680 ± 0.250	99.680 ± 0.250
Autoreasoner width	4.29 ± 1.11	0.42 ± 0.89	85.740 ± 3.650	89.890 ± 3.140
Autoreasoner hybrid	2.78 ± 1.16	0.47 ± 0.50	100.000 ± 0.000	100.000 ± 0.000

NQ depth extension. Table 9 reuses the same $N = 8$ checkpoints at each K. Accuracy scores a single answer; coverage always uses 20 draws without selection. NQ8 largely saturates by $K = 16$, while NQ10 continues to improve. This isolates additional execution from longer training, but does not hold computation fixed across rows.

Table 9: Further NQ execution with training fixed at $N = 8$ (%, mean ± std over seeds). Accuracy uses one answer; coverage uses 20 draws.

	NQ8		NQ10	
K	Accuracy	Coverage	Accuracy	Coverage
8	98.47 ± 0.69	90.44 ± 0.09	84.47 ± 0.70	55.82 ± 0.14
16	$\mathbf{99.95 \pm 0.05}$	$\mathbf{91.55 \pm 0.10}$	91.89 ± 0.54	60.45 ± 0.05
32	99.92 ± 0.08	91.49 ± 0.38	$\mathbf{92.52 \pm 0.17}$	$\mathbf{60.81 \pm 0.14}$

E.5 Training controls and GPU memory

Matched backbone and scoring. All Table 3 conditions are trained separately on the same 39.0M-parameter GC8 backbone, with $N = K = 4$, global batch 768, learning rate 5×10^{-4}, dropout 0, and 34.944M processed transitions (three seeds). The models in Section 5.1 use dropout 0.1 and form a different cohort. Success is the fraction of 255 test graphs whose first unselected answer satisfies both graph constraints and canonical color order; it is neither node accuracy nor coverage.

Supervision and gradient credit. Let $r_t = -\mathbb{E}_{q_t} \log p_\theta(y \mid x, z_t)$ and $\kappa_t = \mathrm{KL}(q_t \| b_t)$. Omitting recipe-specific scaling and free bits, the full-BPTT losses are

$$J_{\text{terminal}} = r_{N-1} + \sum_{t=0}^{N-1} \kappa_t, \qquad J_{\text{repeated}} = \sum_{t=0}^{N-1} (r_t + \kappa_t).$$

Both retain all stage KLs and the complete recurrent graph, with one optimizer update per rollout. Terminal supervision omits intermediate reconstruction terms, not intermediate KLs. These are controls of the same backbone, rather than VRNN reproductions on sequence data.

Accumulated uses repeated local objectives with detached writer inputs, retaining only the immediately preceding write's gradient path. Parameters stay fixed while local gradients accumulate across four reads, followed by one optimizer update. The *Autoreasoner* instead updates after each local backward pass, as in Algorithm 1. Memory values persist across iterations in both cases.

Constant latent keeps recurrence but replaces the writer's sampled z_t with a shared, learned, zero-initialized token; prediction still uses sampled latents. *Fixed memory* bypasses the writer and uses h_0 for every read. Its unselected final draw has the same marginal distribution as one draw from that unchanged prior; it is a VAE-like control under the Autoreasoner's recipe, not the separately trained VAE baseline.

The Sudoku fixed-memory and constant-latent controls use the primary $N = 8$ recipe, the same transition count, and three separately trained seeds. Each is evaluated with 200,000 boards per

seed at $K = 4, 8, 16, 32, 64, 128$, with no valid outputs observed. Their posterior-path failures are described in Appendix E.3.

Memory measurement. Profiling uses one 24 GB RTX 4090, batch 96, bfloat16, no compilation, 40 warm-up and 200 measured steps, and one run per completed cell. Peak allocated GPU memory includes model, optimizer, gradient, and transient tensors; it is not an activation-only estimate. Both full-BPTT conditions fail during the first training iteration at $N = 16$. Failed-run peaks are not treated as completed profiles. These measurements use a fixed batch and are distinct from the global-batch-768 training runs.

Table 10: Peak allocated training memory (GiB), one GPU and batch 96. OOM denotes a failure during training. Allocated GiB use 2^{30} bytes.

Condition	$N = 2$	$N = 4$	$N = 8$	$N = 16$
Terminal + BPTT	4.888	8.945	17.058	OOM
Repeated + BPTT	5.053	9.430	18.195	OOM
Accumulated	3.032	3.030	3.032	3.035
Fixed memory	1.459	1.459	1.459	1.459
Constant latent	2.877	2.877	2.877	2.877
Autoreasoner	2.877	2.877	2.877	2.877

For fixed backbone, state size, and batch, the Autoreasoner retains $O(1)$ activation history in N; our full-BPTT implementation retains $O(N)$ without gradient checkpointing. Checkpointing trades recomputation for lower activation storage. Table 10 concerns memory, not constant training work or a wall-clock speed advantage.

F Model Algorithms

Algorithms 2–5 summarize the model-level computations at the abstraction of Algorithm 1. They omit recipe-specific rescaling, warm-up, and free bits (Appendix E.1); $\mathcal{A}_{\text{recipe}}$ and $\text{StageKL}_{\text{recipe}}$ denote the IAI objective aggregation and HVAE stage balancing. These are schematic descriptions, not complete implementation specifications or reproductions of the cited architectures. Training averages minibatches and uses reparameterized samples. Generation freezes all weights; independent repetitions provide width W.

Algorithm 2 VAE: one-shot variational training and generation

Require: Problem x, training target y, parameters (θ, ϕ)
procedure TrainVAE(x, y)
  $q \leftarrow q_\phi(\cdot \mid x, y)$; $b \leftarrow p_\theta(\cdot \mid x)$; $z \sim q$
  $\mathcal{L} \leftarrow \mathbb{E}_q \log p_\theta(y \mid x, z) - \text{KL}(q \| b)$
  Clear gradients; backpropagate $-\mathcal{L}$; update (θ, ϕ)
end procedure
procedure GenerateVAE(x)
  Freeze parameters; draw $z \sim p_\theta(\cdot \mid x)$
  return $\text{Decode}(p_\theta(\cdot \mid x, z))$
end procedure

Algorithm 3 IAI: iterative posterior inference, one-shot prior generation

Require: Problem x, target y, inference reads R, parameters (θ, ϕ)
procedure TRAINIAI(x, y, R)
  $b \leftarrow p_\theta(\cdot \mid x)$; $\lambda \leftarrow \mathrm{InitPosterior}_\phi(x, y)$
  for $r = 1, \ldots, R$ **do**
    $q_r \leftarrow q(\cdot; \lambda)$; $z_r \sim q_r$
    $\mathcal{L}_r \leftarrow \mathbb{E}_{q_r} \log p_\theta(y \mid x, z) - \mathrm{KL}(q_r \| b)$
    if $r < R$ **then**
      $\lambda \leftarrow \mathrm{Refine}_\phi(x, y, \lambda; \theta)$ ▷ prior b is unchanged
    end if
  end for
  $\mathcal{J} \leftarrow -\mathcal{A}_{\mathrm{recipe}}(\mathcal{L}_1, \ldots, \mathcal{L}_R)$
  Clear gradients; backpropagate $\mathcal{J}$; update (θ, ϕ)
end procedure
procedure GENERATEIAI(x)
  Freeze parameters; draw $z \sim p_\theta(\cdot \mid x)$
  return $\mathrm{Decode}(p_\theta(\cdot \mid x, z))$
end procedure

Algorithm 4 HVAE: hierarchical variational training and ancestral generation

Require: Problem x, target y, layers N, parameters (θ, ϕ)
procedure TRAINHVAE(x, y, N)
  for $\ell = 1, \ldots, N$ **do**
    $c_\ell \leftarrow (x, z_{<\ell})$
    $q_\ell \leftarrow q_{\phi_\ell}(\cdot \mid y, c_\ell)$; $b_\ell \leftarrow p_{\theta_\ell}(\cdot \mid c_\ell)$
    $z_\ell \sim q_\ell$; $\kappa_\ell \leftarrow \mathrm{KL}(q_\ell \| b_\ell)$
  end for
  $\mathcal{L} \leftarrow \mathbb{E}_q \log p_\theta(y \mid x, z_N) - \mathrm{StageKL}_{\mathrm{recipe}}(\kappa_{1:N})$
  Clear gradients; backpropagate $-\mathcal{L}$ through the hierarchy
  Update all layer parameters and the decoder
end procedure
procedure GENERATEHVAE(x)
  Freeze parameters
  for $\ell = 1, \ldots, N$ **do**
    $z_\ell \sim p_{\theta_\ell}(\cdot \mid x, z_{<\ell})$
  end for
  return $\mathrm{Decode}(p_\theta(\cdot \mid x, z_N))$
end procedure

Algorithm 5 Autoreasoner: local training and recursive prior generation

procedure TRAINAUTOREASONER(x, y, N)
  Apply Algorithm 1: repeated local objectives, one-write credit, per-step updates
end procedure
procedure GENERATEAUTOREASONER(x, K)
  Freeze parameters; $h \leftarrow \mathrm{init}(x)$
  for $t = 0, \ldots, K-1$ **do**
    $z \sim p_\theta(\cdot \mid x, h)$
    if $t < K - 1$ **then**
      $h \leftarrow a_\psi(x, h, z)$
    end if
  end for
  return $\mathrm{Decode}(p_\theta(\cdot \mid x, z))$
end procedure