WHAT MATTERS FOR LATENT REASONING WITH FLOW MATCHING

Anonymous authors
Paper under double-blind review

ABSTRACT

Latent reasoning lets a large language model (LLM) think in a continuous space and verbalize only the answer. We argue that an effective latent thought must meet five requirements: it should be *useful*, helping produce the correct answer rather than merely changing it, *diverse*, so that resampling yields different reasoning trajectories, *explainable*, so that a decoded chain of thought (CoT) reflects reasoning the answer actually follows, *refinable* with more inference compute, and *efficient*, costing less than an explicit CoT at comparable accuracy. Current methods rarely meet these requirements: they learn shortcuts from the question, distill the explicit CoT into their weights, or imitate it one token at a time. We focus on flow matching in a learned latent space, the family we argue is best placed to meet them, and identify the training choices that make it work. The result is Flow-based Latent Reasoning (FLaRe), a simple recipe covering what the latent space encodes and how to shape it, where to train the flow, how to read out the answer, and a final stage of training on the model's own verified thoughts. A probe for each requirement shows that FLaRe improves on prior latent methods in all five. It also compares favorably with them on arithmetic benchmarks, while reaching 97% of the accuracy of explicit CoT at a quarter of its latency.

1 INTRODUCTION

Modern large language models (LLMs) increasingly rely on test-time scaling (Snell et al., 2025; DeepSeek-AI et al., 2025; Muennighoff et al., 2025), reasoning at length before producing a final answer. This reasoning is usually written as an explicit chain of thought (CoT) (Wei et al., 2022), generated autoregressively one token at a time. While effective, explicit CoT has three limitations. Its cost grows with the length of the chain, much of this cost is spent on prose that adds little to the reasoning (Hao et al., 2025), and a single rollout commits to its early tokens, so exploring another path requires further generation. Latent reasoning targets these limitations by carrying the intermediate computation in continuous states and verbalizing only the answer. In principle, sampling from a distribution of latent thoughts can explore different trajectories without generating every intermediate step in words, and thus be more efficient.

Existing latent methods differ in how they replace the token-by-token computation of explicit CoT. Horizontal methods feed continuous representations back one slot at a time (Hao et al., 2025; Shen et al., 2025; Tan et al., 2025), while vertical methods reason within hidden states or by looping a block of layers (Deng et al., 2023; 2024; Saunshi et al., 2025; Geiping et al., 2025). Parallel methods refine a block of slots together through fixed-point iterations (Wu et al., 2025; Kuzina et al., 2026) or by denoising in a learned latent space for reasoning (Kang et al., 2026) or text planning (Lovelace et al., 2025). In this work, we argue that an effective latent thought must meet five requirements: it should be useful, diverse, explainable, refinable and efficient. We focus on flow matching in a learned latent space, as in latent diffusion models (Lovelace et al., 2023; Zhang et al., 2023; Kang et al., 2026), since it offers a direct route to four of them. Different noise draws yield different thoughts, a decoder reads each thought back as an explicit CoT, more denoising steps give a thought more compute, and a small code costs less to generate than an explicit CoT. As for usefulness, it depends on the method's design choices, which must ensure that the answer relies on the thought.

Concretely, latent flow matching starts by training a variational autoencoder (VAE) to compress CoTs into a latent space. Conditioned on the question, a flow model then refines Gaussian noise into

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.

a latent thought in this space, from which the same model generates the final answer autoregressively. In practice, with standard text diffusion, this pipeline falls far behind explicit CoT, and it works only when a few training choices are made right. We present these choices as a simple recipe, Flow-based Latent Reasoning (FLaRe), that follows the pipeline from the latent space to the last stage of training (Fig. 1). The VAE, trained with a dual reconstruction objective, encodes a symbolic CoT, stripped of its wording, into a small code that a fine-tuned decoder reads back both as the symbolic CoT and, given the question, as its natural language counterpart. Its latent space is shaped by corrupting the inputs and codes in training rather than by tightening reconstruction. The flow is trained mostly near pure noise, where inference starts, and with several CoTs per question. Its answer reader is trained on the flow's own predicted thoughts as well as on noised codes, so that it learns to read imperfect thoughts as generated at test time. Finally, a short stage 2 self-trains the model on its own verified rollouts for new questions, generating thoughts from pure noise as at inference and backpropagating the answer loss through the full rollout. Our main contributions are:

- **A simple training recipe.** Through controlled ablations, we identify the training choices that make latent flow matching work for reasoning, from learning the latent space to self-training on verified rollouts, and combine them into FLaRe.
- **Requirements for latent reasoning, and probes for them.** We define five requirements an effective latent thought must meet, that it be useful, diverse, explainable, refinable and efficient, design a probe for each, and show that FLaRe improves on prior latent methods in all five.
- **Improved accuracy and efficiency.** FLaRe compares favorably with prior latent methods on arithmetic benchmarks and reaches 97% of explicit CoT accuracy at a quarter of its latency.

2 Preliminaries

2.1 Explicit and Latent Reasoning

First, we provide the relevant background and key concepts of reasoning with LLMs. Given a question q, a model with parameters θ produces a reasoning variable $r \sim p_\theta(r \mid q)$, followed by an answer $a \sim p_\theta(a \mid q, r)$. Explicit and latent reasoning differ in the form of r.

Explicit CoT. With explicit CoT (Wei et al., 2022), $r = (c_1, \ldots, c_L)$ is a sequence of vocabulary tokens generated autoregressively, $p_\theta(r \mid q) = \prod_{l=1}^{L} p_\theta(c_l \mid q, c_{<l})$. We study supervised fine-tuning (SFT) on a dataset $\mathcal{D} = \{(q, r, a)\}$ of questions q, explicit CoTs r and answers a. This explicit CoT training dataset will also be used to train our latent reasoning model.

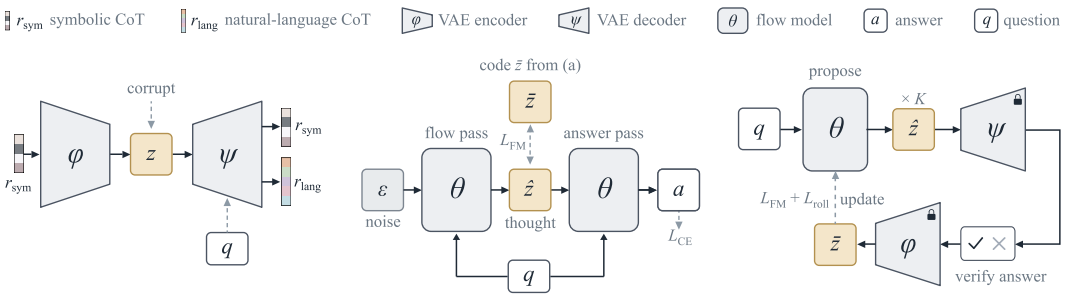
Latent reasoning. Here r stays in a continuous space and is never verbalized during thinking. Vertical methods reason within the model's hidden states without a separate thought. Horizontal and parallel methods, including ours, instead use a latent thought $r = \mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_M) \in \mathbb{R}^{M \times d}$ of M slots of dimension d, with $\mathbf{z} \sim p_\theta(\mathbf{z} \mid q)$. Horizontal methods produce the slots one at a time, while parallel methods refine all of them together. After thinking, the answer is generated token by token from the question and the latent thought, $a \sim p_\theta(a \mid q, \mathbf{z})$.

2.2 Latent Reasoning with Flow Matching

In this paper, we learn the latent thought $\mathbf{z} \in \mathbb{R}^{M \times d}$ via flow matching in a learned latent space, as in latent diffusion (Lovelace et al., 2023; Kang et al., 2026), in two steps. First, a VAE (Kingma & Welling, 2014) is trained on the explicit CoT dataset to encode each CoT into a compressed latent code. Then, a flow model (Lipman et al., 2023) is trained to generate these latent codes conditioned on the question (Kang et al., 2026). At inference, the same shared flow model thinks by denoising, refining Gaussian noise into a latent thought, then produces the final answer autoregressively.

Architecture. The VAE encoder, the VAE decoder and the shared flow model θ, which generates both the latent thought and the answer, are all LLMs (Kang et al., 2026).

VAE training. Both the encoder and the decoder are initialized from a base LLM with parameters ϕ and ψ, respectively. The encoder reads $[r\ \langle \mathrm{Z} \rangle_1 \cdots \langle \mathrm{Z} \rangle_M]$, an explicit CoT r followed by M learned slot tokens. Its final hidden states at the slot positions give the mean and log-variance of the posterior $q_\phi(\mathbf{z} \mid r) = \mathcal{N}\big(\boldsymbol{\mu}_\phi(r), \operatorname{diag} \boldsymbol{\sigma}^2_\phi(r)\big)$, from which the code is sampled as $\mathbf{z} = \boldsymbol{\mu}_\phi(r) + \boldsymbol{\sigma}_\phi(r) \odot \boldsymbol{\epsilon}$ with $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The decoder reads $[\mathbf{z}\ \langle \mathrm{AE} \rangle]$, where $\langle \mathrm{AE} \rangle$ is a learned marker, and reconstructs r



(a) Learning the latent space (b) Stage 1 - Flow matching (c) Stage 2 - Verified self-training

Figure 1: Flow-based Latent Reasoning (FLaRe). (a) A VAE encodes a symbolic CoT r_{sym} into a small code $\mathbf{z}$, corrupted in training for a smooth latent space, and its decoder reads it back as r_{sym} or, given the question q, as a natural language CoT r_{lang}. (b) In stage 1, the shared flow model θ learns to denoise the code given the question and to answer from its own thought $\hat{\mathbf{z}}$. (c) In stage 2, on new questions with answers only, θ proposes K thoughts per question, and those whose decoded answer is correct are re-encoded as flow targets. The model is updated with the flow loss on these targets and an answer loss $\mathcal{L}_{\text{roll}}$ backpropagated through the full denoising path.

from the code alone. The encoder and decoder minimize the β-VAE objective (Higgins et al., 2017):

$$\mathcal{L}_{\text{VAE}} = \mathbb{E}_{q_\phi(\mathbf{z}|r)}\big[-\log p_\psi(r \mid \mathbf{z})\big] + \beta\, D_{\text{KL}}\big(q_\phi(\mathbf{z} \mid r)\,\|\,\mathcal{N}(\mathbf{0},\mathbf{I})\big), \tag{1}$$

where β weights the KL term. After training, the flow targets are the posterior means, standardized per slot and dimension as $\bar{\mathbf{z}} = (\boldsymbol{\mu}_\phi(r) - \mathbf{m})/\mathbf{s}$ (Meshchaninov et al., 2025).

Flow matching. After VAE training, each example becomes $(q, \bar{\mathbf{z}}, a)$. For $t \in [0,1]$, we interpolate from Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ at $t=0$ to the clean VAE code $\bar{\mathbf{z}}$ at $t=1$: $\mathbf{z}_t = (1-t)\boldsymbol{\epsilon} + t\bar{\mathbf{z}}$. The flow model learns the constant velocity $\bar{\mathbf{z}} - \boldsymbol{\epsilon}$ along this path (Lipman et al., 2023; Liu et al., 2023; Li & He, 2026). The shared model θ reads $[q\ \langle\text{SOT}\rangle\ \langle\text{t}\rangle\ \mathbf{z}_t\ \langle\text{EOT}\rangle\ \langle\text{SOA}\rangle\ a]$, where $\langle\text{SOT}\rangle$ and $\langle\text{EOT}\rangle$ bracket the thought, $\langle\text{t}\rangle$ embeds time, and $\langle\text{SOA}\rangle$ starts the answer (Kang et al., 2026). In the flow pass, the M slots attend bidirectionally, and their final hidden states are projected to dimension d to predict velocity under the flow matching loss as follows:

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{(q,\bar{\mathbf{z}}),\,t,\,\boldsymbol{\epsilon}}\Big[\big\|\mathbf{v}_\theta(\mathbf{z}_t, t, q) - (\bar{\mathbf{z}} - \boldsymbol{\epsilon})\big\|_2^2\Big], \tag{2}$$

where t is drawn from a distribution over $[0,1]$ that sets which noise levels receive most of the training signal. We average the loss over four independent $(t, \boldsymbol{\epsilon})$ draws per code, as in MAR (Li et al., 2024), to reduce gradient variance and speed convergence. During flow training, the question is replaced by a null token $\emptyset$ with probability 0.1 to enable classifier-free guidance (CFG) at inference (Ho & Salimans, 2022). After the flow pass, the same training step runs an answer pass, which in its base form reads the clean VAE code $\bar{\mathbf{z}}$ at $t=1$ and minimizes $\mathcal{L}_{\text{CE}} = -\log p_\theta(a \mid q, \bar{\mathbf{z}})$. The joint objective is $\mathcal{L} = \lambda_{\text{FM}}\mathcal{L}_{\text{FM}} + \mathcal{L}_{\text{CE}}$, with λ_{FM} weighting the flow loss.

Inference. Given a question q, we start from Gaussian noise $\mathbf{z}_0 = \boldsymbol{\epsilon}$ and run S Euler steps from $t=0$ to $t=1$ with guidance scale w. The resulting latent thought $\hat{\mathbf{z}} = \mathbf{z}_1$ serves as the reasoning variable, and the same shared model generates the answer $a \sim p_\theta(a \mid q, \hat{\mathbf{z}})$ autoregressively. These steps replace the generation of an explicit CoT, while different noise draws can yield different thoughts for the same question. The frozen VAE decoder renders $\hat{\mathbf{z}}$ as an explicit CoT for offline analysis and for checking decoded final answers against references during stage 2.

2.3 Setup

Default setting. The VAE encoder and decoder are initialized from Llama-3.2-1B and Llama-3.2-3B (Grattafiori et al., 2024), respectively, pretrained on OpenMathInstruct-2 (OMI-2) (Toshniwal et al., 2025), and then trained for 10 epochs on GSM8K-Aug (Deng et al., 2023), an augmentation of GSM8K (Cobbe et al., 2021) with 385K questions and their symbolic CoTs (Fig. 3). The latent thought has $M = 8$ slots of dimension $d = 512$. The flow model is Llama-3.2-1B-Instruct, trained for 40 epochs on the same data with an exponential moving average (EMA) of its weights. Inference uses $S = 20$ Euler steps and a guidance scale $w = 4$. Appendix A.1 lists all hyperparameters.

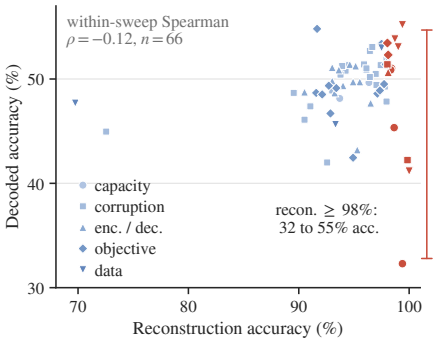


Figure 2: Reconstruction against decoded accuracy for 66 VAEs from nine sweeps.

Table 1: Capacity of the latent code (M slots of dimension d) and size and fine-tuning of the decoder ψ (%).

Latent code $M \times d$	**Recon.**	**Direct**	**Decoded**
32×512	**99.4**	25.4	32.3
16×512	98.6	39.7	45.3
8×256	93.7	42.7	48.1
8×768	96.4	44.4	49.7
8×512	96.8	**47.1**	**53.2**
Decoder ψ	**Recon.**	**Direct**	**Decoded**
1B, frozen	95.3	37.2	43.4
1B, fine-tuned	93.3	40.8	46.7
3B, frozen	96.5	41.9	47.7
3B, fine-tuned	**96.8**	**47.1**	**53.2**

Evaluation. We report GSM8K test accuracy under two readings of the same generated latent thought $\hat{\mathbf{z}}$. **Direct** accuracy uses the answer the shared flow model generates from $\hat{\mathbf{z}}$, the only reading used at inference. **Decoded** accuracy decodes $\hat{\mathbf{z}}$ into a CoT with the frozen VAE decoder and extracts its answer by string matching. The two readings measure different things. The decoded reading checks whether the model thinks correctly, that is, whether its thought solves the question, while the direct reading checks whether the model can then use this thought to answer. The decoded reading is also more forgiving, because the decoder generates the CoT token by token and can turn a latent near a valid code into a well-formed CoT, so it still relies on explicit CoT generation. The direct reading has no such correction and tests how cleanly the thought lands in the VAE space. The gap between the readings thus measures the mismatch between generated thoughts and clean VAE codes that the decoder hides. A thought that decodes to a wrong answer typically fails both readings, while one that decodes to the correct answer can still be misread and fail the direct one.

3 Design Choices and Their Effect

3.1 The Latent Space

The VAE defines the latent space used by the flow model, so we study four choices that shape this space: code capacity, corruption, CoT format and decoder routes, and decoder strength. For each VAE, we report its reconstruction on held-out CoTs, measured by exact text match, and the direct and decoded accuracies (Sec. 2.3) of a flow model trained on its codes.

Table 2: Corruption of the VAE (%).

Corruption	**Recon.**	**Direct**	**Decoded**
None	**99.8**	43.8	49.4
No token substitution	99.6	45.6	47.6
No latent noise	94.2	44.5	50.8
No latent dropout	96.1	44.8	51.0
All three	96.8	**47.1**	**53.2**

Table 3: Encoder input and decoder routes (%). Reconstruction is on the input's own route, if any.

Encoder input, decoder routes	**Recon.**	**Direct**	**Decoded**
r_{lang}, language route only	43.8	40.9	44.7
r_{sym}, language route only	–	46.3	59.8
r_{sym}, symbolic route only	98.5	48.6	55.6
r_{sym}, dual routes	**99.1**	**50.7**	**61.3**

Reconstruction alone is not sufficient. High VAE reconstruction does not by itself produce codes that the flow can generate accurately. At $d = 512$, increasing the code from 8 to 32 slots raises reconstruction from 96.8% to 99.4% but lowers direct accuracy from 47.1% to 25.4% and decoded accuracy from 53.2% to 32.3% (Tab. 1). Across 66 VAEs from nine sweeps that vary the capacity, corruption, objective, encoder, decoder and training data, reconstruction and decoded accuracy have a correlation of -0.12 (Fig. 2), and several of the best-reconstructing VAEs are among the worst downstream. Higher reconstruction thus makes the code more precise but harder to generate. At inference, the flow must build from the question alone a complete thought that solves the problem and remains decodable, so every detail the code preserves is one more detail to generate.

Question Answer: 41

Mike went to a charity event where he bought 4 balloons for five dollars each and 3 water bottles for seven dollars each. What did Mike spend?

Symbolic CoT

<<4*5=20>> <<3*7=21>> <<20+21=41>>

Natural language CoT

Mike spent 4 * 5 = 20 dollars on balloons.
He spent 3 * 7 = 21 dollars on water bottles.
So, Mike spent 20 + 21 = 41 dollars in total.

Figure 3: Symbolic (GSM8K-Aug) and aligned natural language (GSM8K-Aug-NL) CoTs of one question.

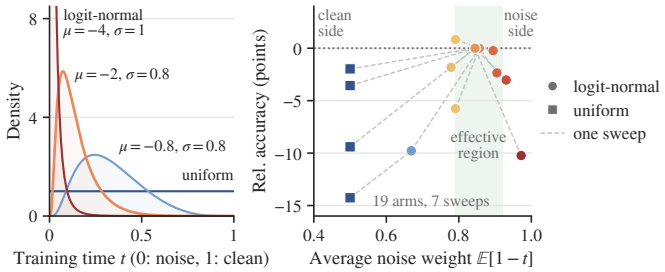


Figure 4: Time distribution of the flow loss. Left, the tested densities of t. Right, accuracy relative to the default of each sweep (gray lines) against the noise weight $\mathbb{E}[1-t]$, effective region in green.

Smoothness is necessary. A generated thought rarely matches a clean VAE code exactly. Nearby codes should therefore decode consistently, and the decoder must tolerate approximate thoughts. Reconstruction and the KL term of equation 1 do not ensure this, and increasing β degrades reconstruction before it improves generation. We thus keep $\beta = 10^{-5}$ and corrupt both the encoder input and latent code during VAE training. Each input CoT token is replaced by a random token with probability $p_{\text{sub}} = 0.3$, while the decoder target stays clean. Half of the sampled codes receive variance-preserving noise, $\mathbf{z} \leftarrow \delta\mathbf{z} + \sqrt{1-\delta^2}\boldsymbol{\xi}$, with $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\delta = 0.7$. We also apply latent dropout with probability $p_{\text{drop}} = 0.4$ (Meshchaninov et al., 2025). All three corruptions improve direct accuracy relative to removing any one of them. Token substitution has the largest effect on decoded accuracy (Tab. 2). Compared with no corruption, they improve both readings despite lower reconstruction. The corruptions thus do the opposite of added capacity: each CoT is encoded as a smooth region rather than a precise point, so a thought that lands nearby still decodes to it, and the flow only has to reach the region. Without them, neighboring codes decode to different CoTs, so a thought that lands near the right code keeps the plan but may be decoded to different values.

The target must be information dense. To see which style of CoT better fits latent reasoning with flow matching, we pair each question in our training data with two CoTs: a symbolic CoT r_{sym} that only records the reasoning steps and intermediate results needed to reach the answer, and its paired natural language CoT r_{lang} that explains the same steps in plain language (Fig. 3). The training set is thus $\mathcal{D}_{\text{vae}} = \{(q, r_{\text{sym}}, r_{\text{lang}}, a)\}$.

The encoder then reads either CoT, and the decoder writes a CoT back through one of two routes. The symbolic route writes r_{sym} from the code alone, $[\mathbf{z}\ \langle\text{AE}\rangle]$, so the code must carry every reasoning step. The language route writes r_{lang}, and with symbolic input it also reads the question, $[\mathbf{z}\ \langle\text{Q}\rangle\ q\ \langle\text{AE}\rangle]$ with a learned marker $\langle\text{Q}\rangle$, which supplies the entities, units and wording that the symbolic CoT omits. Training both routes, which we call the dual routes, gives the following dual reconstruction objective:

$$\mathcal{L}_{\text{VAE}} = \mathbb{E}_{q_\phi(\mathbf{z}|r_{\text{sym}})}\big[-\log p_\psi(r_{\text{sym}} \mid \mathbf{z}) - \log p_\psi(r_{\text{lang}} \mid \mathbf{z}, q)\big] + \beta\, D_{\text{KL}}\big(q_\phi(\mathbf{z} \mid r_{\text{sym}}) \,\|\, \mathcal{N}(\mathbf{0}, \mathbf{I})\big). \quad (3)$$

A VAE with a single route keeps only its own reconstruction term. In Tab. 3 we compare four VAEs with different inputs and decoder routes. The results show that the symbolic CoT is the better input and a VAE that encodes the natural language CoT loses about 10 direct and 17 decoded points against the default. The symbolic CoT is also far more compressible, since the same eight slots reconstruct nearly every held-out symbolic CoT exactly but fewer than half of the natural language ones. Training the decoder on the dual routes also helps in two ways. First, the second reconstruction regularizes the code for more effective flow matching, and second, it yields a single frozen decoder that renders a latent thought as either a symbolic or a natural language CoT, which we use to probe the model. We therefore take the symbolic CoT as input and the dual routes as our default.

For datasets with only natural language CoTs, we propose to convert each CoT with an LLM that, given the symbolic language and its rules, segments the CoT into elementary steps and compiles each step into its symbolic form (Appendix D).

A stronger decoder improves downstream accuracy. With the 1B encoder fixed, a fine-tuned 3B decoder gives the strongest result among the decoder sizes and training choices in Tab. 1. One

possible explanation is that it lets the encoder omit details the decoder can recover, giving the flow an easier target. It may also read approximate generated thoughts more robustly.

The final recipe. We carry forward an 8×512 VAE with symbolic CoTs as encoder input, dual decoder routes, all three corruptions, a 1B encoder, and a fine-tuned 3B decoder.

3.2 The Flow Matching

The flow model is trained on $\mathcal{D}_{\text{flow}} = \{(q, \bar{\mathbf{z}}, a)\}$, where cached VAE codes replace the explicit CoTs of $\mathcal{D}_{\text{vae}}$. Its training objective is $\mathcal{L} = \lambda_{\text{FM}}\mathcal{L}_{\text{FM}} + \mathcal{L}_{\text{CE}}$, as defined in Sec. 2.2, with $\lambda_{\text{FM}} = 5$. We study four choices: where along the path to train, what the answer pass reads, which training data to use, and whether to train end to end through the full rollout.

The flow is trained mostly near noise. The flow loss of equation 2 regresses velocity at a sampled time t, whose distribution determines which parts of the path receive most of the training. Our sweeps vary only this distribution, comparing uniform time with logit-normals (Esser et al., 2024) shifted increasingly toward noise (Fig. 4). Uniform time performs worst, and the model learns well only when training is strongly shifted toward noise, where inference starts. Otherwise, it mainly learns to refine nearly intact codes. The required shift grows with latent dimension (Esser et al., 2024; Guo et al., 2026), since larger codes retain more information at the same noise level. We therefore use a logit-normal time distribution, $t = \text{sigmoid}(s)$ with $s \sim \mathcal{N}(-2, 0.8^2)$.

Table 4: Input of the answer pass in training (%).

Answer pass reads	Direct	Decoded
Exact code	45.7	57.2
Noised code	46.1	**58.0**
50/50 mixture, with gradients	50.2	56.6
Predicted only	51.7	56.4
50/50 mixture	**53.3**	57.3

Table 5: Training data of the flow model, with the VAE fixed (%).

Flow training data	Direct	Decoded
GSM8K-Aug CoT	51.9	56.7
+ Diverse CoT	**54.0**	**59.7**
+ IID data	51.7	58.5
+ OOD data	46.3	52.8
+ IID and OOD data	47.8	55.3

The answer pass must read imperfect thoughts. In each training iteration, a second forward pass through the same model places a latent code $\mathbf{z}_{\text{ans}}$ at the slot positions at time $t = 1$ and predicts the answer tokens with the cross-entropy loss $\mathcal{L}_{\text{CE}} = -\log p_\theta(a \mid q, \mathbf{z}_{\text{ans}})$. Tab. 4 compares five settings. Exact code reads the VAE code $\bar{\mathbf{z}}$, while noised code reads it lightly pulled toward noise. Predicted only reads the flow's one-step endpoint $\hat{\mathbf{z}} = \mathbf{z}_t + (1 - t)\,\tilde{\mathbf{v}}_\theta(\mathbf{z}_t, t, q)$, computed with the guided velocity $\tilde{\mathbf{v}}_\theta$ used at inference and detached from the flow. The two mixture settings read a 50/50 split of predicted endpoints and noised codes, either detaching the endpoints or allowing the answer loss to train the flow through them. The direct reading results in Tab. 4 show that training on the model's own predictions is necessary. Since generated thoughts differ from VAE codes, a reader trained on exact or noised codes alone misreads them, and mixing predictions with noised codes 50/50 works best. However, these predictions must stay detached, since they are noisy one-step estimates of the code, and backpropagating the answer loss through them conflicts with the flow loss.

Table 6: Stage 2 design (%). Each ablation changes one part of the loop.

Training	Direct	Decoded	Targets	Direct	Decoded	Answer loss	Direct	Decoded
Stage 1	55.3	61.3	No verification	57.1	60.7	No answer loss	55.1	60.7
Stage 2	**59.1**	**62.6**	One trace	58.8	61.8	Detached	56.8	60.0
No online flow loss	59.0	61.6	No harvesting	57.2	61.8	1-step rollout	56.3	60.7
No stage 1 replay	59.0	62.5	Raw latents	58.4	62.2	5-step rollout	58.8	61.9

Several CoTs per question help the flow model. With the VAE fixed, we extend the flow training data, which has mostly one CoT per question, in three directions: more solutions per question written by an off-the-shelf LLM (Diverse CoT), new questions of the same style (IID data), and questions of a different style (OOD data). The new questions are the GSM8K-like and MATH-like questions

of OMI-2, whose natural language CoTs we convert into symbolic ones (Appendix D). Tab. 5 shows that several CoTs per question give the largest gain in both readings, while new questions help less and questions of a different style lower accuracy. With one CoT per question, different noise draws still yield different thoughts, but each question is paired with a single target code, and diffusion models trained on few samples per condition tend to memorize them (Gu et al., 2025; Kadkhodaie et al., 2024). With several CoTs, each question has several modes, and the flow must map different noise draws to different valid solutions rather than to one code. Diverse training data is thus necessary for latent reasoning with flow matching, as it teaches the flow to cover the space of all valid solutions rather than a single one.

End-to-end training through the full rollout is required. After stage 1, the decoded reading remains several points above the direct one. The generated thoughts therefore often contain the correct answer, but the model's readout is not adapted to them. Two limitations keep stage 1 from closing this gap. First, the answer pass trains on a mixture of noised codes and detached one-step endpoints, while inference builds thoughts from pure noise over S guided steps, whose errors differ from those seen in training. Second, the answer loss updates the shared backbone but does not train thought generation through the flow steps. Addressing both requires generating thoughts from pure noise as at inference and backpropagating the answer loss through the full rollout.

In our recipe, we keep this simple and propose a stage 2 of self-training on the model's own verified rollouts, following STaR and ReST-EM (Zelikman et al., 2022; Singh et al., 2024). This stage uses only new questions and their reference answers, without requiring additional CoT annotations, a learned reward or a reinforcement learning objective. For each block of new questions, the EMA model generates $K = 8$ thoughts per question from centered noise over $S = 20$ guided steps. The frozen VAE decoder reads each thought into a symbolic CoT, and all distinct CoTs whose final answer matches the reference are retained. The retained CoTs are re-encoded into latent codes for the next round of training updates. Each update combines the flow loss on these codes and stage 1 replay with an answer loss on a fresh rollout from noise over $S = 10$ guided steps. The answer loss backpropagates through every step and both guidance branches. This teaches the reader to use generated thoughts and the flow to produce thoughts that yield the reference answer. The updated EMA model then generates thoughts for the next block.

Tab. 6 examines each part of this loop with the parent model, VAE, question pool and schedule held fixed. Stage 2 improves the direct reading by 3.8 points and the decoded reading by 1.3. These gains come from the answer loss through the full rollout, since removing it, detaching the rollout endpoint or reducing the rollout to a single step loses most or all of them. Verification is also needed, while the other choices matter less. Appendix A.3 details the implementation and discusses each ablation.

The final recipe. We carry forward a flow model trained on GSM8K-Aug and Diverse CoT, using a logit-normal time distribution shifted toward noise. In stage 1, the answer pass reads a 50/50 mixture of noised codes and the model's own detached endpoints. A short stage 2 then self-trains on verified thoughts for about one pass over a pool of new questions.

4 Analysis: The Requirements of Latent Reasoning

Effective latent reasoning requires thoughts that are **useful**, helping produce the correct answer, and **diverse**, so that resampling can explore different reasoning trajectories for the same question. These trajectories should also be **explainable**, with recovered CoTs that reflect the reasoning the answer actually follows. At inference, the thoughts should remain **refinable** with more compute and **efficient**, costing less than explicit CoT at comparable accuracy. With our final recipe fixed, we examine these requirements through targeted probes, comparing FLaRe with Coconut, CODI and PCCoT, with explicit CoT as a reference. Tab. 7 summarizes the results, with details in Appendix E.

Useful latent thoughts. A latent thought should carry question-specific computation that helps produce the correct answer. For Coconut and CODI, prior work reports little change in accuracy when thoughts are removed, especially on logical reasoning tasks (Rizvi-Martel et al., 2026; Dilgren & Wiegreffe, 2026), or, for Coconut, swapped across questions (Zhang et al., 2025a). Training by curriculum or distillation can internalize the CoT in the model's weights, allowing a shortcut from the question to the answer (Zhang et al., 2025a; Cui et al., 2026; Aswal et al., 2026). We probe this dependence with a *twin test*: changing one number in a question produces a twin with a different

Table 7: The five requirements on GSM8K. Useful: twin pairs (%) whose answer follows the injected thought. Diverse: gain over greedy with 16 samples. Explainable: questions (%) whose decoded thought has all reference intermediate results in order. Refinable: gain from 0.1 to 0.5× explicit CoT compute. Efficient: speedup over explicit CoT at the share of its accuracy. FLaRe: decoded reading, except refinable (direct / decoded) and efficient (direct, $S = 2$).

	Useful	**Diverse**		**Explainable**	**Refinable**	**Efficient**
Method	follows	pass@16	vote	values in order	0.1 → 0.5× CoT compute	speedup at % CoT acc.
Explicit CoT	97.2	–	–	–	–	1.0× at 100%
Coconut	7.3	+4.6	−1.5	19.1	+2.8	2.6× at 61%
CODI	31.8	+3.5	−0.7	38.5	+6.1	2.0× at 94%
PCCoT	27.7	+0.8	−1.9	7.8	−10.5	3.9× at 91%
FLaRe, stage 1	39.7	+16.6	+3.3	46.0	+0.4 / +18.2	3.9× at 93%
FLaRe, stage 2	37.3	+14.7	+1.8	48.6	+1.8 / +24.2	3.9× at 97%

answer. We inject the model's thought for the twin into the original question and measure how often the answer matches the twin's answer. Explicit CoT follows the twin's CoT on 97% of pairs, compared with 40% and 37% for our two stages, 32% for CODI, 28% for PCCoT and 7% for Coconut (Tab. 7). The harder twins often produce incorrect thoughts. Among pairs that each model answers correctly on both questions before injection, our following rates rise to 91% in stage 1 and 76% in stage 2. Zeroing the thought or giving the reader only the question also reduces accuracy in both stages, supporting the thoughts' contribution to correct answers (Appendix E.1).

Diverse latent thoughts. Resampling should explore different valid trajectories for the same question, providing coverage that voting, verification, reinforcement learning or self-training can turn into higher accuracy. Coconut, CODI and PCCoT generate a single thought deterministically, and the superposition Coconut was designed for collapses after fine-tuning (Rizvi-Martel et al., 2026). New candidates therefore require perturbing the thought at inference (Wang et al., 2026b;a). We probe this diversity with a *sampling test* of 16 thoughts per question, using independent noise draws for our flow and noise perturbations for the baselines. We measure coverage as pass@16, the share of questions solved by at least one sample, and test whether majority voting improves on the greedy answer. The baselines gain at most 5 points of coverage, and voting never improves on greedy. On the decoded reading, our stage 1 model gains 16.6 points of coverage and 3.3 through voting, while stage 2 gains 14.7 and 1.8, respectively. Stage 2 retains most of this coverage while improving single-sample accuracy (Appendix E.2).

Explainable latent thoughts. An explainable latent thought should decode into an explicit CoT that reflects the reasoning the answer follows. The baselines lack a dedicated thought decoder, so their thoughts are read through the LM head, since a thought fed back as an embedding often behaves like its nearest token (Rizvi-Martel et al., 2026). This recovers only fragments (Shen et al., 2025), and full trajectories require an explicit search over candidate traces (Dilgren & Wiegreffe, 2026), which still does not show that the answer follows them (Aswal et al., 2026). We probe recovery with a *reading test* that checks whether every reference intermediate result appears in order, in the top-1 LM-head tokens for the baselines and as complete values in the CoT decoded by the frozen VAE decoder for ours. Our thoughts meet this criterion on about half of the questions, more often than any baseline (Tab. 7). We then probe faithfulness on twin pairs whose edit changes an intermediate value. Our decoded thoughts usually recover the twin's value, and the answer follows this recovered reasoning far more often than for the baselines (Appendix E.3).

Refinable latent thoughts. More inference compute should refine a latent thought until accuracy saturates, as in test-time scaling of explicit CoT (Snell et al., 2025; Muennighoff et al., 2025). Existing methods are constrained by their training budgets: Coconut becomes unstable beyond two thoughts per reasoning step (Hao et al., 2025), CODI peaks at six latent tokens (Shen et al., 2025), and PCCoT at three Jacobi iterations (Wu et al., 2025). We probe this with a *budget sweep* of each frozen model, including budgets beyond those used in training. In Tab. 7, raising the budget from a tenth to half that of explicit CoT adds at most 6.1 points to the baselines, and PCCoT even loses accuracy, while our decoded reading gains 18.2 and 24.2 points in stages 1 and 2 and the direct reading gains at most 1.8. Since the flow is trained across noise levels, more Euler steps keep refining its thought, while the baselines peak at or below their training budgets (Appendix E.4).

Table 8: Direct answer accuracy (%) of models trained on GSM8K-Aug, on GSM8K (IID) and on GSM8K-Hard, SVAMP and MultiArith (OOD). Bold: best latent method per column and scale. Baselines come from the original papers when reported, else from our runs (Appendix A.5).

	Qwen2.5-0.5B-Instruct				Llama-3.2-1B-Instruct				Llama-3.2-3B-Instruct			
	IID	OOD			IID	OOD			IID	OOD		
Method	GSM8K	Hard	SVAMP	MArith	GSM8K	Hard	SVAMP	MArith	GSM8K	Hard	SVAMP	MArith
Explicit CoT	59.7	17.0	61.6	98.3	62.5	14.7	66.5	97.8	72.4	20.8	73.4	98.3
No-CoT	27.3	6.4	38.5	57.8	35.0	7.4	38.1	67.8	39.2	9.3	56.7	96.7
Coconut	28.2	6.8	41.9	58.3	36.1	8.3	48.4	87.2	45.3	10.9	57.5	96.7
iCoT	32.9	7.4	41.0	66.7	36.5	8.4	40.8	72.8	42.1	9.6	50.7	91.1
LaDiR	39.3	9.0	44.5	88.3	40.6	8.9	38.6	81.7	41.5	10.2	49.4	94.4
PCCoT	30.6	7.3	36.3	67.2	53.4	12.8	57.5	97.2	57.4	13.8	70.9	**100.0**
CODI	39.1	9.1	46.9	**90.0**	55.6	12.8	**61.1**	96.1	59.5	14.9	71.2	99.4
KaVa	46.9	10.8	50.6	–	56.5	12.7	58.9	–	**65.7**	**15.2**	**72.7**	–
FLaRe, stage 1	50.0	11.7	49.1	78.3	55.3	12.5	57.0	94.4	62.5	14.8	63.8	98.3
FLaRe, stage 2	**53.8**	**13.0**	**51.0**	88.3	**59.1**	**13.7**	58.8	**98.3**	65.2	**15.2**	66.2	99.4

Efficient latent thoughts. Since latent reasoning does not verbalize every step, it should cost less than explicit CoT at comparable accuracy. We probe this trade-off with a *latency test* of accuracy and time to answer on one GPU, one question at a time. In Tab. 7, the best baselines reach 94% of explicit CoT accuracy at a 2.0× speedup (CODI) or 91% at 3.9× (PCCoT), while our stage 2 model, with the direct reading at two Euler steps, reaches 97% at 3.9× (Appendix E.5).

5 Comparison with Prior Latent Reasoning Methods

Following CODI and KaVa, Tab. 8 reports direct accuracy on GSM8K and zero-shot transfer to GSM8K-Hard (Gao et al., 2023), SVAMP (Patel et al., 2021) and MultiArith (Roy & Roth, 2015). FLaRe follows the final recipe of Sec. 3.2, whose Diverse CoT and stage 2 questions help adapt the LLM to flow matching, a task it was not pretrained for, while all other methods except LaDiR build on its autoregressive abilities. Even stage 1, trained on the same GSM8K-Aug and Diverse CoT rows as LaDiR, our most direct competitor, which also denoises in a learned latent space, gains 10.7 to 21.0 points on GSM8K, so the gap comes from the design choices of Sec. 3 rather than from the data. Stage 2 then improves every column at every scale, by 2.5 to 3.8 points on GSM8K. It leads all latent methods on GSM8K at 0.5B and 1B, by 6.9 and 2.6 points over KaVa, performs comparably to it at 3B, and leads or ties on GSM8K-Hard at every scale. SVAMP is the main exception, likely because nearly half of its questions add distractor numbers, which are rare in GSM8K-Aug.

6 Related Work

Implicit CoT moves explicit reasoning (Wei et al., 2022) into hidden states (Deng et al., 2023; 2024), while filler tokens add computation without feedback (Goyal et al., 2024; Pfau et al., 2024). Continuous feedback methods generate latent slots sequentially (Hao et al., 2025; Shen et al., 2025; Tan et al., 2025), and looped and parallel methods refine hidden states iteratively (Saunshi et al., 2025; Geiping et al., 2025; Wu et al., 2025; Kuzina et al., 2026). Most methods learn from explicit CoT through a curriculum, SFT or distillation. We build on latent diffusion, which denoises codes in a learned latent space (Lovelace et al., 2023; Kang et al., 2026; Lovelace et al., 2025), and identify the training choices that make it work for reasoning. Appendix F compares these families.

7 Conclusion

In this paper, we studied what matters for latent reasoning with flow matching through controlled ablations of every stage of the pipeline, and distilled the answers into a simple recipe. We then set out five requirements a latent reasoning method should meet, useful, diverse, explainable, refinable and efficient, designed a probe for each, and found that the thoughts of FLaRe improve on those of prior latent methods in all five. FLaRe also compares favorably with them on arithmetic benchmarks, while reaching 97% of the accuracy of explicit CoT at a quarter of its latency.

AI USE STATEMENT

We used generative AI tools to generate the synthetic data of Appendices C and D, which is part of the method, and, alongside the authors, to implement the method, design experiments and interpret their results. We did not use them to develop the ideas of the paper or its hypotheses, and no mathematical claims, proofs, translation or qualitative analysis were involved. Additionally, we used them to draft parts of the appendix, refine the main text for readability, create figures, edit code, and search the literature. We have reviewed all AI-assisted work, testing the code and checking every number, claim and reference against its source, and we take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work trains and evaluates language models on public grade-school and competition math datasets, GSM8K and its augmentations and OpenMathInstruct-2, and involves no human subjects, personal data or sensitive content. The symbolic CoTs we generate are derived from these public sources with open-weight models, and we keep only those consistent with the reference answers. Latent reasoning removes the explicit CoT at inference, which reduces what can be monitored, so our recipe keeps every thought decodable through the frozen VAE decoder, and Sec. 4 measures how faithful that decoding is. We see no other ethical concerns beyond those common to training language models for reasoning.

REPRODUCIBILITY STATEMENT

Appendix A lists every hyperparameter of the final recipe and the evaluation protocol (Tab. 9), and details the answer pass of stage 1 and the stage 2 loop with their equations. Appendix B gives the algorithms of training and inference, Appendix C describes the datasets and the exact corpora with their sizes, Appendix D the symbolic language and the conversion pipeline, and Appendix A.3 the stage 2 question pool. Appendix E gives the protocol of every probe of Sec. 4. The training code, the data pipeline, the probes, the converted corpora and the checkpoints will be released with the paper.

ACKNOWLEDGMENTS

N/A

REFERENCES

Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning. In *Conference on Language Modeling*, 2025.

Darpan Aswal, Thomas Palmeira Ferraz, Yongxin Zhou, and Maxime Peyrard. Observable patterns are not explanations: A causal-geometric analysis of latent reasoning models. *arXiv preprint arXiv:2606.12689*, 2026.

Roman Bachmann, Jesse Allardice, David Mizrahi, Enrico Fini, Oğuzhan Fatih Kar, Elmira Amirloo, Alaaeldin El-Nouby, Amir Zamir, and Afshin Dehghan. FlexTok: Resampling images into 1d token sequences of flexible length. In *International Conference on Machine Learning*, 2025.

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

Eden Biran, Daniela Gottesman, Sohee Yang, Mor Geva, and Amir Globerson. Hopping too late: Exploring the limitations of large language models on multi-hop queries. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024.

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

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023.

Xinghao Chen, Anhao Zhao, Heming Xia, Xuan Lu, Hanlin Wang, Yanjun Chen, Wei Zhang, Jian Wang, Wenjie Li, and Xiaoyu Shen. Reasoning beyond language: A comprehensive survey on latent chain-of-thought reasoning. *arXiv preprint arXiv:2505.16782*, 2025.

Jeffrey Cheng and Benjamin Van Durme. Compressed chain of thought: Efficient reasoning through dense representations. *arXiv preprint arXiv:2412.13171*, 2024.

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

Yingqian Cui, Zhenwei Dai, Bing He, Zhan Shi, Hui Liu, Rui Sun, Zhiji Liu, Yue Xing, Jiliang Tang, and Benoit Dumoulin. How do latent reasoning methods perform under weak and strong supervision? *arXiv preprint arXiv:2602.22441*, 2026.

Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers. In *International Conference on Learning Representations*, 2024.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

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

Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. Implicit chain of thought reasoning via knowledge distillation. *arXiv preprint arXiv:2311.01460*, 2023.

Yuntian Deng, Yejin Choi, and Stuart Shieber. From explicit CoT to implicit CoT: Learning to internalize CoT step by step. *arXiv preprint arXiv:2405.14838*, 2024.

Connor Dilgren and Sarah Wiegreffe. Are latent reasoning models easily interpretable? *arXiv preprint arXiv:2604.04902*, 2026.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Lacey, Alex Goodwin, Yannik Marek, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis. In *International Conference on Machine Learning*, 2024.

Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization. In *International Conference on Learning Representations*, 2025.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *International Conference on Machine Learning*, 2023.

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

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

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. In *International Conference on Learning Representations*, 2024.

Aaron Grattafiori et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Xiangming Gu, Chao Du, Tianyu Pang, Chongxuan Li, Min Lin, and Ye Wang. On memorization in diffusion models. *Transactions on Machine Learning Research*, 2025.

Hongcan Guo, Qinyu Zhao, Yian Zhao, Shen Nie, Rui Zhu, Qiushan Guo, Feng Wang, Tao Yang, Hengshuang Zhao, Guoqiang Wei, and Yan Zeng. Continuous latent diffusion language model. *arXiv preprint arXiv:2605.06548*, 2026.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. In *Second Conference on Language Modeling*, 2025.

David Herel and Tomas Mikolov. Thinking tokens for language modeling. *arXiv preprint arXiv:2405.08644*, 2024.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv preprint arXiv:2510.04871*, 2025.

Zahra Kadkhodaie, Florentin Guth, Eero P. Simoncelli, and Stéphane Mallat. Generalization in diffusion models arises from geometry-adaptive harmonic representations. In *International Conference on Learning Representations*, 2024.

Haoqiang Kang, Yizhe Zhang, Nikki Lijing Kuang, Nicklas Majamaki, Navdeep Jaitly, Yi-An Ma, and Lianhui Qin. LaDiR: Latent diffusion enhances LLMs for text reasoning. In *International Conference on Learning Representations*, 2026.

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations*, 2014.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, and Ali Farhadi. Matryoshka representation learning. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Anna Kuzina, Maciej Pióro, and Babak Ehteshami Bejnordi. KaVa: Latent reasoning via compressed KV-cache distillation. In *International Conference on Learning Representations*, 2026.

Tianhong Li and Kaiming He. Back to basics: Let denoising generative models denoise. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

Tianhong Li, Yonglong Tian, He Li, Mingyang Deng, and Kaiming He. Autoregressive image generation without vector quantization. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Jack Lindsey, Wes Gurnee, Emmanuel Ameisen, Brian Chen, Adam Pearce, Nicholas L. Turner, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, et al. On the biology of a large language model. *Transformer Circuits Thread*, 2025. URL https://transformer-circuits.pub/2025/attribution-graphs/biology.html.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *International Conference on Learning Representations*, 2023.

Justin Lovelace, Varsha Kishore, Chao Wan, Eliot Shekhtman, and Kilian Q Weinberger. Latent diffusion for language generation. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Justin Lovelace, Christian Belardi, Sofian Zalouk, Adhitya Polavaram, Srivatsa Kundurthy, and Kilian Q Weinberger. Stop-Think-AutoRegress: Language modeling with latent diffusion planning. In *Conference on Language Modeling*, 2025.

Matthew V. Macfarlane, Clément Bonnet, Herke van Hoof, and Levi H. S. Lelis. Gradient-based program synthesis with neurally interpreted languages. In *International Conference on Learning Representations*, 2026.

Sean McLeish, Ang Li, John Kirchenbauer, Dayal Singh Kalra, Brian R. Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Jonas Geiping, Tom Goldstein, and Micah Goldblum. Teaching pretrained language models to think deeper with retrofitted recurrence. *arXiv preprint arXiv:2511.07384*, 2025.

Viacheslav Meshchaninov, Egor Chimbulatov, Alexander Shabalin, Aleksandr Abramov, and Dmitry Vetrov. Cosmos: Compressed and smooth latent space for text diffusion modeling. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321, 2025.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021.

Jacob Pfau, William Merrill, and Samuel R Bowman. Let's think dot by dot: Hidden computation in transformer language models. In *First Conference on Language Modeling*, 2024.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026. URL https://qwen.ai/blog?id=qwen3.8.

Oren Rippel, Michael A. Gelbart, and Ryan P. Adams. Learning ordered representations with nested dropout. In *International Conference on Machine Learning*, 2014.

Michael Rizvi-Martel, Guillaume Rabusseau, and Marius Mosbach. The illusion of superposition? A principled analysis of latent thinking in language models. *arXiv preprint arXiv:2604.06374*, 2026.

Subhro Roy and Dan Roth. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015.

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

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. CODI: Compressing chain-of-thought into continuous space via self-distillation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 677–693, 2025.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, Abhishek Kumar, Alex Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura Culp, Lechao Xiao, Maxwell L. Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yundi Qian, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024.

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

Wenhui Tan, Jiaze Li, Jianzhong Ju, Zhenbo Luo, Ruihua Song, and Jian Luan. Think silently, think fast: Dynamic latent compression of LLM reasoning chains. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. OpenMathInstruct-2: Accelerating AI for math with massive open-source instruction data. In *International Conference on Learning Representations*, 2025.

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.

Minghan Wang, Ye Bai, Thuy-Trang Vu, Ehsan Shareghi, and Gholamreza Haffari. GTS: Inference-time scaling of latent reasoning with a learnable Gaussian thought sampler. *arXiv preprint arXiv:2602.14077*, 2026a.

Minghan Wang, Thuy-Trang Vu, Ehsan Shareghi, and Gholamreza Haffari. Towards inference-time scaling for continuous space reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 26842–26856, 2026b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022.

Xilin Wei, Xiaoran Liu, Yuhang Zang, Xiaoyi Dong, Yuhang Cao, Jiaqi Wang, Xipeng Qiu, and Dahua Lin. SIM-CoT: Supervised implicit chain-of-thought. In *International Conference on Learning Representations*, 2026.

Haoyi Wu, Zhihao Teng, and Kewei Tu. Parallel continuous chain-of-thought with Jacobi iteration. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 914–926, 2025.

Wilson Yan, Volodymyr Mnih, Aleksandra Faust, Matei Zaharia, Pieter Abbeel, and Hao Liu. ElasticTok: Adaptive tokenization for image and video. In *International Conference on Learning Representations*, 2025.

An Yang et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024a.

Sohee Yang, Elena Gribovskaya, Nora Kassner, Mor Geva, and Sebastian Riedel. Do large language models latently perform multi-hop reasoning? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024b.

Ping Yu, Jing Xu, Jason Weston, and Ilia Kulikov. Distilling system 2 into system 1. *arXiv preprint arXiv:2407.06023*, 2024.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, 2022.

Yizhe Zhang, Jiatao Gu, Zhuofeng Wu, Shuangfei Zhai, Joshua Susskind, and Navdeep Jaitly. PLANNER: Generating diversified paragraph via latent language diffusion model. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Yuyi Zhang, Boyu Tang, Tianjie Ju, Sufeng Duan, and Gongshen Liu. Do latent tokens think? A causal and adversarial analysis of chain-of-continuous-thought. *arXiv preprint arXiv:2512.21711*, 2025a.

Zhen Zhang, Xuehai He, Weixiang Yan, Ao Shen, Chenyang Zhao, and Xin Eric Wang. Soft thinking: Unlocking the reasoning potential of LLMs in continuous concept space. In *Advances in Neural Information Processing Systems*, volume 38, 2025b.

Hanlin Zhu, Shibo Hao, Zhiting Hu, Jiantao Jiao, Stuart Russell, and Yuandong Tian. Reasoning by superposition: A theoretical perspective on chain of continuous thought. In *Advances in Neural Information Processing Systems*, volume 38, 2025a.

Rui-Jie Zhu, Tianhao Peng, Tianhao Cheng, Xingwei Qu, Jinfa Huang, Dawei Zhu, Hao Wang, Kaiwen Xue, Xuanliang Zhang, Yong Shan, et al. A survey on latent reasoning. *arXiv preprint arXiv:2507.06203*, 2025b.

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

APPENDIX

A Method and Experimental Details . . . 16
A.1 Implementation Details. . . . 16
A.2 Flow Model, Stage 1 . . . 16
A.3 Flow Model, Stage 2 . . . 18
A.4 Variable-Length Latent Thoughts . . . 20
A.5 Baseline Numbers . . . 20
A.6 LaDiR Baseline . . . 21
B Algorithms . . . 22
C Datasets . . . 23
D Symbolic Conversion Pipeline . . . 24
E The Requirements of Latent Reasoning: Details . . . 24
E.1 Useful: The Twin Test . . . 25
E.2 Diverse: Sampling by Noise . . . 25
E.3 Explainable: Reading the Thoughts . . . 26
E.4 Refinable: Accuracy against Inference Budget . . . 27
E.5 Efficient: Latency Measurement . . . 27
F Extended Related Work . . . 28

A METHOD AND EXPERIMENTAL DETAILS

A.1 IMPLEMENTATION DETAILS

Tab. 9 lists the full details and hyperparameters of the final recipe.

A.2 FLOW MODEL, STAGE 1

A.2.1 ANSWER PASS

Two inputs. Each training row contributes one answer pass in which the slot positions hold a latent $\mathbf{z}_{\text{ans}}$ at time $t = 1$. The row uses the model's own one-step endpoint with probability $p_{\text{pred}} = 0.5$ and the noised code otherwise,

$$\mathbf{z}_{\text{ans}} = \begin{cases} \hat{\mathbf{z}} & \text{if } u < p_{\text{pred}}, \\ (1-\rho)\,\bar{\mathbf{z}} + \rho\,\boldsymbol{\epsilon} & \text{otherwise,} \end{cases} \qquad u \sim \mathcal{U}[0,1]. \tag{4}$$

Noised code. When the answer pass takes the cached code as input, the code is pulled toward noise with $\rho \sim \mathcal{U}[0, \rho_{\max}]$, $\rho_{\max} = 0.3$ and $\boldsymbol{\epsilon}$ the first noise draw of the flow loss for that row. This is done to simulate the imperfect thoughts the model generates at inference.

One-step endpoint. When the answer pass takes the model's own thought as input, $\mathbf{z}_{\text{ans}} = \hat{\mathbf{z}}$ is the one-step estimate of the clean code under classifier-free guidance,

$$\begin{aligned} \hat{\mathbf{z}} &= \mathbf{z}_t + (1-t)\,\tilde{\mathbf{v}}_\theta(\mathbf{z}_t, t, q), \\ \tilde{\mathbf{v}}_\theta(\mathbf{z}_t, t, q) &= \mathbf{v}_\theta(\mathbf{z}_t, t, \emptyset) + w\left[\mathbf{v}_\theta(\mathbf{z}_t, t, q) - \mathbf{v}_\theta(\mathbf{z}_t, t, \emptyset)\right], \end{aligned} \tag{5}$$

computed without gradient from the first flow draw $(\mathbf{z}_t, t)$ with scale $w = 4$. Since the target velocity is constant along the path, $\hat{\mathbf{z}}$ is the current estimate of the clean code from any intermediate state, and mirrors the thoughts produced at inference. Note that it costs two extra forward passes per row, one conditioned and one with the null question.

Question masking. The answer pass always sees the question, and the null-token replacement for classifier-free guidance applies to the flow pass only.

A.2.2 NOISE DRAWS PER CODE

Instead of a single noise draw per example, the flow loss of equation 2 is computed over n independent draws of $(t, \boldsymbol{\epsilon})$ for the same latent code and averaged as in MAR (Li et al., 2024). Fig. 5 tracks the accuracy of the EMA and LIVE weights over training for n from 1 to 8. With EMA weights,

Table 9: Implementation details of the final recipe: the VAE, the flow model of stages 1 and 2, the inference and the evaluation protocol.

VAE

The VAE is initialized from a model trained for one epoch on the natural language CoTs of OMI-2 (Toshniwal et al., 2025) with 32 slots and a single natural language route. It is then reduced to $M = 8$ slots at initialization and trained with the dual objective of equation 3 with both terms weighted equally. Token substitution draws the replacement uniformly from the vocabulary with clean decoder targets, and the latent noise and dropout act on the sampled code before decoding. The statistics $(\mathbf{m}, \mathbf{s})$ are the mean and standard deviation of the posterior means of 20K training rows.

Backbones ϕ, ψ	Llama-3.2-1B / 3B, fine-tuned	Token substitution p_{sub}	0.3 on the encoder input
Initialization	OMI-2 language VAE, 32×512	Latent noise	VP, $\delta = 0.7$, on half of the codes
Latent code $M \times d$	8×512	Latent dropout p_{drop}	0.4
Input, decoder routes	r_{sym}, dual routes, equal weights	Optimizer	AdamW, $(0.9, 0.98)$, wd 10^{-5}, clip 1.0
Training data	paired GSM8K-Aug, 367K rows	Learning rate	10^{-4} encoder, $2 \cdot 10^{-5}$ decoder
Epochs, batch, sequence	10, 256, 512 tokens	Schedule	cosine, 5% warmup
KL weight β, free bits	10^{-5}, 2.0 nats	Statistics $(\mathbf{m}, \mathbf{s})$	per slot and dimension, 20K rows

Flow model, stage 1

The flow and answer passes map a latent into the hidden space of θ through separate projectors, a LayerNorm followed by a two-layer MLP per slot, with a slot embedding shared by both passes. The time enters as a sinusoidal embedding at the $\langle \text{t} \rangle$ position and through an adaptive LayerNorm in the zero-initialized velocity head. The learning rate decays with a cosine over the last fifth of training, and the EMA is a single-precision shadow updated after every step.

Backbone θ	Llama-3.2-1B-Instruct	EMA decay	0.9999
Input sequence	$[q \ \langle \text{SOT} \rangle \ \langle \text{t} \rangle \ \mathbf{z}_t \ \langle \text{EOT} \rangle \ \langle \text{SOA} \rangle \ a]$	Loss weights	$\lambda_{\text{FM}} = 5$, answer loss 1
Prediction target	velocity $\bar{\mathbf{z}} - \boldsymbol{\epsilon}$	Time distribution	logit-normal, $s \sim \mathcal{N}(-2, 0.8^2)$
Latent projectors	separate for the two passes	Draws per code n	4, independent $(t, \boldsymbol{\epsilon})$
Slot, time embedding	shared by both passes	Question drop (CFG)	0.1
Training data	GSM8K-Aug 385K + Diverse CoT 216K rows	Answer pass input	noised code or detached endpoint
Training updates	60K	$\rho_{\max}$, endpoint w	0.3, 4
Batch, sequence	256, 512 tokens	Endpoint share p_{pred}	0.5
Optimizer	AdamW, $(0.9, 0.95)$, wd 10^{-5}, clip 1.0	Hardware	16 GPUs, FSDP, bf16
Learning rate	10^{-4}, warmup-stable-decay	Seed	42
Warmup, decay	first 1,000, last 20% of the updates		

Flow model, stage 2

The loop of stage 2 runs in blocks: every B updates, the EMA proposer draws K thoughts for each question of a new block, the decoded CoTs whose final value matches the reference are re-encoded with the frozen VAE, and the next B updates minimize the flow loss on these codes, mixed with stage 1 replay, together with the answer loss backpropagated through the full rollout from noise. The proposer uses the EMA weights, and the final stage 2 model is the LIVE one.

Question pool	100K OMI-2 questions, answers only	Replay α, draws n	0.25, 4 sharing one noise
Block N, updates B	1,024 questions, 16 updates	Rollout	$S_{\text{train}} = 10$, $w = 4$, all steps
Proposals K, proposer	8, EMA weights (decay 0.999)	Loss weights	$\lambda_{\text{FM}} = 5$, answer loss 1
Proposal sampler	Euler, $S = 20$, $w = 4$, noise 1.0	Optimizer	AdamW, $(0.9, 0.95)$, wd 10^{-5}, clip 1.0
Verification	decoded final value = reference	Learning rate	$3.5 \cdot 10^{-6}$, cosine, 1% warmup
Kept per question	all distinct accepted, at most K	Updates	1,600 (100 blocks)
Targets	re-encoded posterior means	Precision	fp32 weights, bf16 autocast, DDP
Batch per update	64 block rows, 16 replay rows	Final model	LIVE weights, not the EMA

Inference

The initial noise is seeded by the evaluation seed and the question index, so every model sees the same noise, and is integrated with the guided Euler update of equation 6. The same model then generates the answer greedily from $\hat{\mathbf{z}}$ with the question visible, and the decoded reading maps $\hat{\mathbf{z}}$ back with $(\mathbf{m}, \mathbf{s})$ and decodes it greedily with the frozen decoder, without the question.

Solver	Euler, $S = 20$, uniform grid	Direct reading	greedy, at most 64 tokens
Guidance w, noise scale	4, 1.0	Decoded reading	frozen ψ, greedy, at most 512 tokens
Initial noise	one canonical seed per question		

Evaluation

Accuracies are over the full test sets with one generation per question, the prediction being the last number of the answer or of the decoded CoT, correct when it equals the reference within 10^{-4}. Stage 1 is evaluated with the EMA weights and stage 2 with the LIVE weights. The OOD evaluation sets run zero-shot with the same prompt, sampler, seed and grader as GSM8K: GSM8K-Hard (Gao et al., 2023), SVAMP (Patel et al., 2021) and MultiArith (Roy & Roth, 2015).

Answer extraction	last number, match at 10^{-4}	Stage 1 weights	EMA
Test sets	GSM8K 1,319, GSM8K-Hard 1,319 SVAMP 1,000, MultiArith 180	Stage 2 weights	LIVE

the final accuracies remain within about one point across all numbers of draws on both readings. Multiple draws mainly improve convergence speed. With a single draw, the EMA weights trail the other three by 2 to 4 points on the direct reading from epoch 8 to 20 and on the decoded reading at epoch 12, catching up by epochs 28 and 20, respectively, while two, four and eight draws stay within about two points of each other throughout. The LIVE weights show the same gap earlier, with a single draw 5 to 6 points behind on the direct reading and 3 to 6 on the decoded one at epoch 4, closing by epoch 20, and they end 7 to 10 points below the EMA weights. However, note that

each draw costs one more flow pass per row, so that eight draws cost about 1.6 times the runtime of four, and we keep $n = 4$.

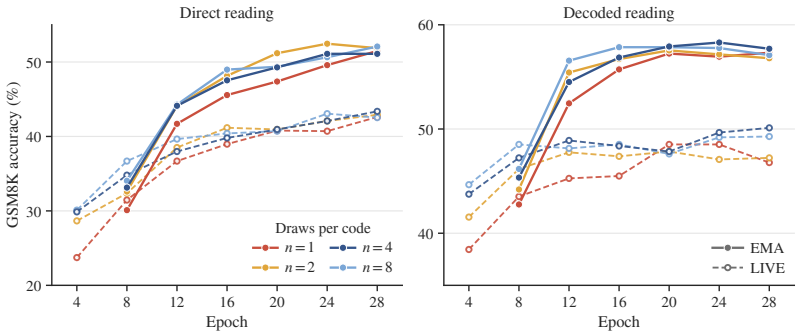


Figure 5: Noise draws per code. GSM8K accuracy (%) over training for one, two, four and eight draws per code, under the direct and decoded readings, for the EMA weights (solid, filled markers) and the LIVE weights (dashed, open markers).

A.3 Flow Model, Stage 2

Stage 2 is the self-training loop of Sec. 3.2 run in blocks of $B = 16$ updates. At the start of each block, the EMA model proposes K thoughts for each of the N new questions, and the frozen decoder converts each thought into a symbolic CoT. CoTs whose final answer matches the reference are retained and re-encoded by the VAE. The next B updates use these codes as flow targets and backpropagate an answer loss through a fresh rollout. The block is then replaced by the next one and the loop cycles over the question pool. We describe each step in the order of the loop next.

Question pool. The pool holds 100K questions of OMI-2 (Toshniwal et al., 2025) with their reference answers and no CoT, disjoint from $\mathcal{D}_{\text{flow}}$ and from the GSM8K test set. A seeded permutation of the pool is split across the 16 GPUs, and each block takes the next $N = 1{,}024$ questions, 64 per GPU. The permutation wraps around, so a question left unsolved is proposed again in the next pass.

Proposals. The proposer is the EMA of the trained weights (decay 0.999) updated after every optimizer step. For each question, it draws $K = 8$ Gaussian samples $\boldsymbol{\eta}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and centers and rescales them into the starts

$$\boldsymbol{\epsilon}_k = \sqrt{\frac{K}{K-1}} \left(\boldsymbol{\eta}_k - \frac{1}{K} \sum_{j=1}^{K} \boldsymbol{\eta}_j \right),$$

so that each start is still a standard Gaussian while the K starts sum to zero. Each start is then integrated as at inference, with $S = 20$ Euler steps on the uniform grid $t_s = s/S$ and the guided velocity of scale $w = 4$,

$$\begin{aligned} \tilde{\mathbf{v}}_\theta(\mathbf{z}, t, q) &= \mathbf{v}_\theta(\mathbf{z}, t, \emptyset) + w \left[\mathbf{v}_\theta(\mathbf{z}, t, q) - \mathbf{v}_\theta(\mathbf{z}, t, \emptyset) \right], \\ \mathbf{z}^{(s+1)} &= \mathbf{z}^{(s)} + (t_{s+1} - t_s)\, \tilde{\mathbf{v}}_\theta\big(\mathbf{z}^{(s)}, t_s, q\big), \end{aligned} \tag{6}$$

from $\mathbf{z}^{(0)} = \boldsymbol{\epsilon}_k$ at unit noise scale, to the thought $\hat{\mathbf{z}}_k = \mathbf{z}^{(S)}$, and we write this map as $G_\theta^S(q, \boldsymbol{\epsilon}) = \hat{\mathbf{z}}$. Proposals use it with the EMA weights and without gradient, and the answer loss below uses it with the LIVE weights and with gradients.

Why centered starts. Independent draws share a random common offset which pushes all K thoughts of a question in the same direction. Centering removes this offset, and the rescaling keeps each start an exact sample of the Gaussian prior so a single proposal is as accurate as before. In a matched pilot on 10,000 training questions with $K = 16$, centered and independent starts give the same share of correct candidates (93.1%), but centered starts raise the median number of distinct decoded CoTs per question from one to two and find a new verified CoT for 5.3% more questions.

Verification and targets. Each thought is un-standardized and decoded greedily by the frozen decoder into a symbolic CoT, without the question and with at most 160 new tokens. The CoT is

accepted when its final answer parses and matches the reference under a strict grader. Verification thus only compares the final answer of the decoded CoT with the reference answer, rather than the answer of the direct reading, and it does not certify the intermediate steps. The accepted CoTs of a question are deduplicated on their step text and all distinct ones are kept, while unsolved questions are dropped. Each kept CoT r is re-encoded by the frozen encoder into its posterior mean and standardized with the unchanged stage 1 statistics, $\bar{\mathbf{z}} = (\boldsymbol{\mu}_\phi(r) - \mathbf{m})/\mathbf{s}$. These codes are then used as flow targets and form the training block $\mathcal{B} = \{(q, \bar{\mathbf{z}}, a)\}$. At the start of stage 2 with the EMA stage 1 parent, 65 to 66% of the questions of a block are solved with 2.5 to 2.7 distinct CoTs each.

Batches. Each of the next $B = 16$ updates takes 4 rows per GPU from $\mathcal{B}$. Each GPU goes through its accepted questions in a shuffled order, reshuffling when they are exhausted. Each update also adds one replay row per GPU drawn uniformly with replacement from $\mathcal{D}_{\text{flow}}$ and encoded on the fly. After B updates, the block is replaced even if some of its CoTs were not used.

Losses. The flow loss of equation 2 is computed separately on the online rows of $\mathcal{B}$ and on the replay rows. The two are then mixed as

$$\mathcal{L}_{\text{FM}}^{\text{S2}} = (1-\alpha)\,\mathcal{L}_{\text{FM}}^{\mathcal{B}} + \alpha\,\mathcal{L}_{\text{FM}}^{\mathcal{D}_{\text{flow}}}, \qquad \alpha = 0.25. \tag{7}$$

The answer loss reads a thought generated by the LIVE weights from fresh noise over $S_{\text{train}} = 10$ guided Euler steps of equation 6, with gradients through every step and both guidance branches,

$$\mathcal{L}_{\text{roll}} = \mathbb{E}_{(q,a)\sim\mathcal{B},\,\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\Big[-\log p_\theta\big(a \mid q, G_\theta^{S_{\text{train}}}(q,\boldsymbol{\epsilon})\big)\Big], \tag{8}$$

where the thought sits at the slot positions at $t = 1$ with the question, as in stage 1, and a is the reference answer. This rollout starts from new independent noise and not from the proposal starts. The loss is averaged over the answer tokens and is not applied to replay rows. The objective is

$$\mathcal{L}_{\text{Stage 2}} = \lambda_{\text{FM}}\,\mathcal{L}_{\text{FM}}^{\text{S2}} + \mathcal{L}_{\text{roll}}, \qquad \lambda_{\text{FM}} = 5, \tag{9}$$

in which each term has one role: the online flow loss reproduces the verified thoughts, the replay term keeps the flow close to the stage 1 data, and the answer loss is the only term backpropagated through the full rollout.

Optimization and evaluation. Stage 2 starts from the EMA weights of the stage 1 parent with a fresh AdamW optimizer ($\epsilon = 10^{-6}$, other settings in Tab. 9) and runs for 1,600 updates with seed 42. This corresponds to 100 blocks and 100K question visits, approximately one pass over the pool, though not over every retained CoT. Unlike stage 1, the final stage 2 model is the LIVE model, not the EMA. On 16 H100 GPUs, a block takes 130 to 160 s to propose and verify and an update 4 to 5 s, about 6 hours per run.

Ablations. The ablations of Tab. 6 keep everything else fixed and change one part of the loop: the targets, the flow loss or the answer loss, as follows.

- *No verification* drops only the answer check: parseable CoTs are still deduplicated, and the answer loss still uses the reference answer rather than the possibly wrong decoded one.
- *One trace* keeps a single accepted CoT per question instead of all distinct ones.
- *Raw latents* keeps verification and deduplication but trains on the first accepted generated endpoint of each distinct CoT instead of its re-encoded code.
- *No online flow loss* removes only $(1-\alpha)\mathcal{L}_{\text{FM}}^{\mathcal{B}}$ and keeps the answer loss and the replay term $\lambda_{\text{FM}}\alpha\mathcal{L}_{\text{FM}}^{\mathcal{D}_{\text{flow}}}$ unchanged to isolate the contribution of the verified flow targets.
- *No stage 1 replay* sets $\alpha = 0$ so the online flow loss takes the full weight.
- *No answer loss* removes $\mathcal{L}_{\text{roll}}$ and keeps both the online flow loss and the replay.
- *Detached* keeps the answer loss but stops its gradient at the generated thought to isolate the effect of backpropagating the gradients through the full flow rollout.
- *1-step rollout* and *5-step rollout* generate the thought of $\mathcal{L}_{\text{roll}}$ with $S_{\text{train}} = 1$ or 5 guided steps instead of 10.

Results. Stage 2 improves the direct reading by 3.8 points and the decoded reading by 1.3 over stage 1 (Tab. 6), and these gains come from the answer loss through the full rollout. Without the answer loss, the model stays at the stage 1 level. Detaching the rollout endpoint loses 2.3 points on the direct reading and 2.6 on the decoded one, and a one-step rollout loses 2.8 and 1.9. Verification

is also needed, as training on unverified thoughts loses 2.0 and 1.9 points. Raw generated latent targets, one verified trace per question, no online flow loss, no stage 1 replay and a five-step rollout all stay within one point of stage 2 on both readings.

A.4 Variable-Length Latent Thoughts

A fixed width asks one code to hold both a one-step and a long CoT, while explicit CoT spends more tokens on harder questions. Our recipe supports a latent thought of variable width with almost no change, and unlike LaDiR (Kang et al., 2026), which varies the length by generating one block after another, the thought stays a single block of slots refined in parallel and only gets wider or narrower. This keeps the method as efficient as with a fixed width since the number of sequential passes stays $2S$ whatever the width, whereas block-wise generation repeats the whole denoising for every block. The VAE registers the largest number of slot tokens and uses the first M of them, the flow model generates M slots with the loss of equation 2 averaged over the active slots, and M can be chosen at inference.

Two designs. The VAE has 16 slot tokens and encodes each CoT with 8, 12 or 16 of them, chosen by the length of the CoT: 8 up to 16 encoder tokens, 12 up to 24, and 16 beyond. In a quarter of the cases one of the two other widths is used instead, so that every width is trained on CoTs of every length, and the flow model is trained on the same mix of widths. The two designs differ in how the codes of different widths relate. In the *nested* design, each slot token attends only to the slots before it, so the first 8 slots of a 16-slot code are exactly the 8-slot code, and one encoder pass at 16 slots gives all three widths, as in Matryoshka representations (Rippel et al., 2014; Kusupati et al., 2022) and nested tokenizers (Bachmann et al., 2025). In the *elastic* design, the slot tokens attend to each other in both directions, so each width has its own code, computed by a separate encoder pass per width, as in elastic tokenizers (Yan et al., 2025).

Outcome on GSM8K-Aug. The nested design trains and generates at every width as intended, but on GSM8K-Aug it brings no gain since 8 slots already suffice. The symbolic CoTs are short, with a median of 21 encoder tokens and 90% below 35 tokens. Eight slots already reconstruct 98.7% of the test CoTs exactly, and increasing the number of slots to 16 adds only 0.4 points. The extra slots therefore carry little, a weak residual of the first eight at about half their scale. As a result, the best variable-width model only matches the fixed width of 8 (60.1 decoded for both), and choosing the width by the length of the CoT does not beat it. While variable widths do not help on this dataset, we expect them to on data whose CoTs vary more in length, since the recipe is flexible and applies to variable widths with almost no change. In that case, the width becomes a thinking budget chosen at inference (Muennighoff et al., 2025; Aggarwal & Welleck, 2025).

A.5 Baseline Numbers

For each baseline entry in Tab. 8, we use the result from the original paper when it reports the relevant backbone and test set. Otherwise, we use our own run or evaluate released weights, leaving the entry empty when neither is available. Tab. 10 lists the sources. We do not copy rows from a single reproduction, since reproductions of one baseline differ by up to nine points, e.g., explicit CoT on Qwen2.5-0.5B-Instruct reaches 50.6 in Kuzina et al. (2026) and 59.7 in our run.

Table 10: Source of each baseline number of Tab. 8. P: printed by the paper that introduced the method, CODI (Shen et al., 2025), PCCoT (Wu et al., 2025) or KaVa (Kuzina et al., 2026). T: trained by us, with LaDiR reimplemented as in Appendix A.6. R: released weights evaluated by us, the public explicit CoT checkpoint `yingfanbot/gsm-cot-llama3b` (a), and those of Dilgren & Wiegreffe (2026) (b) and Wu et al. (2025) (c).

Method	Qwen2.5-0.5B-Instruct	Llama-3.2-1B-Instruct	Llama-3.2-3B-Instruct
Explicit CoT	T	T	R (a)
No-CoT	T	T	T
iCoT	T	T	T
Coconut	T	R (b)	T
CODI	T	P	T
PCCoT	T	P on GSM8K, R (c) on the OOD sets	T
KaVa	P, no MultiArith	P, no MultiArith	P, no MultiArith
LaDiR	T	T	T

Table 11: Recipes of our baseline runs on GSM8K-Aug. Learning rates are for Qwen2.5-0.5B / Llama-3.2-1B / Llama-3.2-3B.

Method	Tuning	Learning rate	Epochs	Other settings
Explicit CoT	LoRA, $r = 128$, $\alpha = 32$	$8 \cdot 10^{-4}$ / $8 \cdot 10^{-4}$ / –	10	batch 128, cosine, 3% warmup, wd 0.1
No-CoT	LoRA, $r = 128$, $\alpha = 32$	$2 \cdot 10^{-4}$ / $8 \cdot 10^{-4}$ / $2 \cdot 10^{-4}$	10	as explicit CoT, answer only
iCoT	full, single precision	10^{-5}	20	batch 32, 8 CoT tokens removed per epoch (Qwen: 11)
Coconut	full, bf16	$5 \cdot 10^{-5}$	3 + 6 + 1	$c = 1$, a CoT stage, stages 1 to 6 and a fully latent stage, batch 128
CODI	LoRA, $r = 128$, $\alpha = 32$	$8 \cdot 10^{-4}$ / – / $3 \cdot 10^{-4}$	10 / – / 8	6 latents, distillation weight 20, batch 128
PCCoT	LoRA, $r = 128$, $\alpha = 32$	$5 \cdot 10^{-4}$ / – / $2 \cdot 10^{-4}$	10	24 latents, $T = 3$ iterations, batch 128

Our runs. We train each method with its official code patched only for Qwen2 support (PCCoT and Coconut) and for multi-GPU training and greedy decoding (iCoT), with the recipes of Tab. 11. Where the original paper has no recipe for a backbone, we follow the closest reproduction: Kuzina et al. (2026) for the LoRA baselines and PCCoT, Wei et al. (2026) for CODI at 3B, and the 1B recipe of Dilgren & Wiegreffe (2026) for Coconut.

Evaluation. We evaluate both our runs and released weights with the same protocol as our model (Tab. 9), using greedy decoding on the full test sets and matching the last number in each output to the reference answer. This protocol agrees with that of the original papers: on the released 1B checkpoints, we obtain 55.6 on GSM8K for CODI against the printed 55.6, and 54.1 for PCCoT against 53.35.

A.6 LaDiR Baseline

We reimplement LaDiR (Kang et al., 2026) as a variable-length, block-autoregressive latent baseline, trained on the same GSM8K-Aug and Diverse CoT rows as our model.

VAE. Each symbolic step takes the place of a sentence and is encoded into its own block, a code of three slots of dimension 512 (so that, averaged over the dataset, a CoT takes about as many slots as our fixed 8). In the last block we include an answer-only suffix so that a stopping head can learn when the CoT is complete. The encoder and decoder (Llama-3.2-1B and Llama-3.2-3B) start from the OMI-2 language VAE that also initializes ours, and the encoder is fine-tuned, on single steps, with the objective of equation 1, with token substitution and additive Gaussian latent noise.

Flow model. The flow model uses the same backbones as ours, Llama-3.2-1B-Instruct, Llama-3.2-3B-Instruct or Qwen2.5-0.5B-Instruct. In stage 1 it predicts the velocity of each noised block given the clean preceding blocks, with uniform time and bidirectional attention within each block, and the answer and stopping losses read the clean codes. Stage 2 keeps the same supervised rows but replaces the preceding blocks with blocks generated by 10-step guided rollouts, with gradients through the rollout. The flow, answer and stopping losses are weighted 5, 1 and 2, and the optimizer is that of our recipe (Tab. 9).

Inference. Blocks are generated sequentially until the stopping head fires or 16 blocks have been generated, using guidance $w = 4$ and noise scale 2. The answer is then sampled at temperature 0.7. As for our model, each block takes 20 Euler steps, and stage 1 is read with the EMA weights and stage 2 with the LIVE weights.

Efficiency comparison. LaDiR generates one block per reasoning step, and each block needs a full denoising of S guided steps, as much as our whole thought. Its cost therefore grows with the length of the reasoning. GSM8K test CoTs have 3.3 steps on average and 5 at the 90th percentile, so at 20 steps LaDiR runs about 130 sequential forward passes per question, against 40 for our model. At our measured 22 ms per guided Euler step, this is about 1.5 s per question on average and 2.2 s at the 90th percentile, against 0.46 s for our direct reading, or 3.2 to 4.8 times slower than our model and 5.5 times slower than explicit CoT. The gap widens with fewer steps. LaDiR needs 10 to 20 Euler steps per block, since its answer is not trained on the model's own one-step endpoints and each block builds on the previously generated ones, whose errors accumulate. Our answer pass is trained to read such endpoints in stage 1, and our direct reading reaches 57.4 after only two steps, in 68 ms, 11 to 21 times faster than LaDiR at 10 to 20 steps per block.

B ALGORITHMS

Algorithms 1, 2 and 3 outline stage 1, stage 2 and inference. The VAE, trained beforehand with equation 3, stays frozen, $\tilde{\mathbf{v}}_\theta(\mathbf{z}, t, q) = \mathbf{v}_\theta(\mathbf{z}, t, \emptyset) + w\,[\mathbf{v}_\theta(\mathbf{z}, t, q) - \mathbf{v}_\theta(\mathbf{z}, t, \emptyset)]$ is the guided velocity with scale w, and $\mathrm{sg}[\cdot]$ stops the gradient.

Algorithm 1 Stage 1 training of the flow model

Input: data $\mathcal{D} = \{(q, r_{\mathrm{sym}}, a)\}$, frozen VAE encoder ϕ with statistics $(\mathbf{m}, \mathbf{s})$, flow model θ

repeat
- sample $(q, r_{\mathrm{sym}}, a) \sim \mathcal{D}$
- $\bar{\mathbf{z}} \leftarrow (\boldsymbol{\mu}_\phi(r_{\mathrm{sym}}) - \mathbf{m})/\mathbf{s}$ ▷ encode the CoT
- $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $t \sim$ logit-normal$(-2, 0.8^2)$ ▷ shifted toward noise
- $\mathbf{z}_t \leftarrow (1-t)\,\boldsymbol{\epsilon} + t\,\bar{\mathbf{z}}$ ▷ point on the straight path from noise to code
- $q' \leftarrow \emptyset$ with probability 0.1, else q ▷ null question for CFG
- $\mathcal{L}_{\mathrm{FM}} \leftarrow \left\|\mathbf{v}_\theta(\mathbf{z}_t, t, q') - (\bar{\mathbf{z}} - \boldsymbol{\epsilon})\right\|_2^2$ ▷ flow pass
- **if** with probability 0.5 **then** ▷ input of the answer pass, 50/50
  - $\mathbf{z}_{\mathrm{ans}} \leftarrow \mathrm{sg}\left[\mathbf{z}_t + (1-t)\,\tilde{\mathbf{v}}_\theta(\mathbf{z}_t, t, q)\right]$ ▷ detached one-step endpoint
- **else**
  - $\mathbf{z}_{\mathrm{ans}} \leftarrow (1-\rho)\,\bar{\mathbf{z}} + \rho\,\boldsymbol{\epsilon}$, $\rho \sim \mathcal{U}[0, 0.3]$ ▷ noised code
- $\mathcal{L}_{\mathrm{CE}} \leftarrow -\log p_\theta(a \mid q, \mathbf{z}_{\mathrm{ans}})$ ▷ answer pass
- gradient step on $\lambda_{\mathrm{FM}}\mathcal{L}_{\mathrm{FM}} + \mathcal{L}_{\mathrm{CE}}$, then update the EMA $\bar{\theta}$

until the end of training

Output: EMA weights $\bar{\theta}$

Algorithm 2 Stage 2 training of the flow model

Input: stage 1 EMA weights θ and data $\mathcal{D}$, frozen VAE (ϕ, ψ) with $(\mathbf{m}, \mathbf{s})$, question pool $\mathcal{Q} = \{(q, a)\}$

$\bar{\theta} \leftarrow \theta$ ▷ EMA proposer

for each block of N new questions from $\mathcal{Q}$ **do**
- **for** each (q, a) in the block, K times **do** ▷ propose and verify, without gradient
  - $\hat{\mathbf{z}} \leftarrow \textsc{Think}(\bar{\theta}, q, S)$ ▷ Algorithm 3
  - decode $\hat{r}$ from $p_\psi(\cdot \mid \mathbf{m} + \mathbf{s} \odot \hat{\mathbf{z}})$ ▷ symbolic route
  - **if** the final answer of $\hat{r}$ is a **then** ▷ verification
    - add $\left(q,\ (\boldsymbol{\mu}_\phi(\hat{r}) - \mathbf{m})/\mathbf{s},\ a\right)$ to the targets $\mathcal{B}$ of the block ▷ re-encode as a flow target
- **for** B updates **do** ▷ train on the block
  - sample $(q, \bar{\mathbf{z}}, a) \sim \mathcal{B}$ and a replay row from $\mathcal{D}$
  - $\mathcal{L}_{\mathrm{FM}} \leftarrow (1-\alpha)\,\mathcal{L}_{\mathrm{FM}}^{\mathcal{B}} + \alpha\,\mathcal{L}_{\mathrm{FM}}^{\mathcal{D}}$ ▷ flow loss of Algorithm 1 on each row
  - $\hat{\mathbf{z}} \leftarrow \textsc{Think}(\theta, q, S_{\mathrm{train}})$ ▷ rollout from fresh noise, with gradients
  - $\mathcal{L}_{\mathrm{roll}} \leftarrow -\log p_\theta(a \mid q, \hat{\mathbf{z}})$ ▷ answer loss through the full rollout
  - gradient step on $\lambda_{\mathrm{FM}}\mathcal{L}_{\mathrm{FM}} + \mathcal{L}_{\mathrm{roll}}$, then update the EMA $\bar{\theta}$

Output: LIVE weights θ

Algorithm 3 Inference, with the decoded CoTs as an optional probe

Input: question q, flow model θ, Euler steps S, guidance scale w, VAE decoder ψ with $(\mathbf{m}, \mathbf{s})$

function $\textsc{Think}(\theta, q, S)$
- $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ▷ start from noise at $t = 0$
- **for** $i = 0, \ldots, S-1$ **do**
  - $\mathbf{z} \leftarrow \mathbf{z} + \tilde{\mathbf{v}}_\theta(\mathbf{z}, i/S, q)/S$ ▷ guided Euler step
- **return** $\mathbf{z}$ ▷ latent thought at $t = 1$

$\hat{\mathbf{z}} \leftarrow \textsc{Think}(\theta, q, S)$

generate a from $p_\theta(\cdot \mid q, \hat{\mathbf{z}})$ ▷ answer, the direct reading

Optional probe: read the thought back as an explicit CoT

$\mathbf{z} \leftarrow \mathbf{m} + \mathbf{s} \odot \hat{\mathbf{z}}$ ▷ undo the standardization

decode $\hat{r}_{\mathrm{sym}}$ from $p_\psi(\cdot \mid \mathbf{z})$ ▷ symbolic route, the decoded reading

decode $\hat{r}_{\mathrm{lang}}$ from $p_\psi(\cdot \mid \mathbf{z}, q)$ ▷ language route, given the question

Output: answer a, and optionally the CoTs $\hat{r}_{\mathrm{sym}}$ and $\hat{r}_{\mathrm{lang}}$

C DATASETS

Tab. 12 lists the corpora used in the paper.

Table 12: Corpora used in the paper. Rows are (question, CoT, answer) triples and questions are distinct normalized question texts.

Corpus	Origin	Rows	Questions
GSM8K-Aug, flow	`whynlp/gsm8k-aug` train	385K	385K
GSM8K-Aug, paired	aligned with `whynlp/gsm8k-aug-nl`	367K	367K
Diverse CoT	Qwen2.5-32B-Instruct on GSM8K-Aug questions	216K	148K
Diverse CoT, paired	same, with a verified natural language twin	205K	142K
IID data	OMI-2 `augmented_gsm8k`, converted	166K	39K
OOD data	OMI-2 `augmented_math`, converted	463K	80K
Stage 2 pool	OMI-2 questions and answers, no CoT	100K	100K

GSM8K-Aug and GSM8K-Aug-NL. GSM8K-Aug (Deng et al., 2023) has 385K training rows, nearly one per distinct question, 500 validation rows and the 1,319 GSM8K test questions (Cobbe et al., 2021). Its CoT is symbolic, one marker `<<expression=value>>` per step, such as `<<600*30/100=180>>`, and the last value is the answer. GSM8K-Aug-NL writes the same steps of the same questions as sentences. We pair the two releases row by row into $\mathcal{D}_{\text{vae}}$ and keep for the VAE the 367K training pairs whose symbolic chain ends at the answer, whose last sentence contains it, and whose intermediate values all appear in the sentences.

Test sets. We evaluate on the 1,319 GSM8K test questions and, zero-shot, on GSM8K-Hard (Gao et al., 2023), the same 1,319 questions with larger numbers, on the 1,000 questions of SVAMP (Patel et al., 2021), and on the 180 test questions of MultiArith (Roy & Roth, 2015).

Diverse CoT. Qwen2.5-32B-Instruct (Yang et al., 2024a) wrote 16 solutions per GSM8K-Aug question directly in the GSM8K-Aug format. A solution is kept when it parses, computes correctly, ends at the reference answer, differs from the stored solution and passes two semantic judges, which leaves 216K CoTs over 148K questions.

IID and OOD data. Both come from OpenMathInstruct-2 (OMI-2) (Toshniwal et al., 2025), whose CoTs are in natural language and are converted to symbolic CoTs by the pipeline of Appendix D. IID data is its `augmented_gsm8k` share and OOD data its `augmented_math` share, with no question overlapping any split of GSM8K-Aug. The VAE data of Tab. 13 adds the paired rows of one source to $\mathcal{D}_{\text{vae}}$ (572K, 533K, 830K and 996K rows), and the flow data of Tab. 5 adds its symbolic rows to the 385K GSM8K-Aug rows (602K, 551K, 849K and 1,015K rows).

Training data of the VAE. The default VAE encodes the symbolic CoTs of GSM8K-Aug, which are of limited diversity with mostly one CoT per question. We thus ask whether a more diverse corpus induces a better latent space, with the flow data held fixed, by adding the Diverse CoT, the IID data or the OOD data above. As shown in Tab. 13, more data can help but only to a very limited degree, since reconstruction is already saturated on the default corpus.

Table 13: Training data of the VAE, with the flow data fixed. Reconstruction (%) and GSM8K accuracy (%) under the direct and decoded readings.

VAE training data	Recon.	Direct	Decoded
GSM8K-Aug CoT	99.1	50.7	56.9
+ Diverse CoT	99.0	48.8	54.8
+ IID data	99.0	**51.6**	56.9
+ OOD data	99.3	50.9	56.0
+ IID and OOD data (5× budget)	**99.6**	51.1	**58.3**

Stage 2 pool. The 100K OMI-2 questions of Appendix A.3 are 73K `augmented_gsm8k` questions and 26K `augmented_math` questions with their reference answers only and no CoT.

D SYMBOLIC CONVERSION PIPELINE

This pipeline turns a dataset with natural language CoTs into the paired format of $\mathcal{D}_{\text{vae}}$.

The symbolic language. The converter targets a small-step language that includes GSM8K-Aug arithmetic as a subset. Each CoT consists of `<<BODY>>` markers in ASCII mathematics, with one move per marker: an equation, an inequality, a declaration `<<x in R>>`, a definition `<<u_1:=3*4>>`, or a guarded case. Every name must come from the question or an earlier declaration, and the final marker contains only the answer.

Converter. Qwen3.8-27B (Qwen Team, 2026), sampled at temperature 1 with its highest reasoning effort and a fixed seed per row, is given the language and its rules, the question, the CoT split into numbered sentences, and the reference answer. It returns groups of elementary symbolic steps, each naming the sentences it compiles, or rejects the CoT, or flags the question as ill-posed, and the natural language CoT is never rewritten.

Run on OMI-2. Before conversion, a filter keeps 2M of the 13M rows of OMI-2. It rejects rows with a false equation or a conclusion that contradicts the label, keeps CoTs under 256 tokens with explicit reasoning and no backtracking, removes the questions of all GSM8K-Aug sets, and keeps at most eight distinct solutions per question. A blind audit, in which the converter answers each question alone, then keeps the 57.5% of the questions whose answer matches the label. The converter succeeds on 96.3% of the remaining rows, and a review in a fresh context, which sees the question, the original CoT and the symbolic one together, accepts 90.5% of the 700,131 distinct traces. After deduplication and a final leakage check, 629,824 pairs over 119,835 questions remain.

Examples. Tab. 14 shows two published pairs, one IID and one OOD.

Table 14: Two published pairs, one IID and one OOD, with the natural language CoT verbatim from OMI-2 and the symbolic CoT written by the converter.

IID pair, from the `augmented_gsm8k` share	
Question	Lucy has orange trees that produce 4 oranges each. If an orange can be sold for $2 and Lucy wants to earn $80, how many orange trees does she need to harvest?
r_{lang}	Each orange tree produces 4 oranges. Each orange can be sold for $2. So the amount of money Lucy earns from each tree is 4 * $2 = $8. If Lucy wants to earn $80, then she needs to harvest $80 / $8 = 10 trees. Thus, Lucy needs to harvest 10 orange trees.
r_{sym}	`<<4*2=8>> <<80/8=10>> <<10>>`
a	10
OOD pair, from the `augmented_math` share	
Question	The greatest common divisor of a, 12, and 21 is 3. What is the smallest possible value of a that is greater than 3?
r_{lang}	The greatest common divisor (GCD) of 12 and 21 is 3. Since a shares the same GCD, a must be a multiple of 3. The smallest multiple of 3 greater than 3 is 6. Therefore, the smallest possible value of a that is greater than 3 is 6.
r_{sym}	`<<a in Z>> <<gcd(12,21)=3 and gcd(a,12,21)=3=>rem(a,3)=0>> <<a>3 and rem(a,3)=0=>a>=6>> <<gcd(6,12,21)=3>> <<6>>`
a	6

Future directions. While we provide this conversion as a simple offline LLM-based step, the converter could also be trained with execution feedback by writing symbolic CoTs as executable programs (Gao et al., 2023; Chen et al., 2023), and the symbolic structure could even be induced directly in the latent space from task feedback alone (Macfarlane et al., 2026). We leave both directions for future work.

E THE REQUIREMENTS OF LATENT REASONING: DETAILS

All probes use Llama-3.2-1B-Instruct fine-tuned on GSM8K-Aug and, unless noted, the 1,319 GSM8K test questions. The baselines are the released checkpoints of CODI (Shen et al., 2025) and PCCoT (Wu et al., 2025), and those of Coconut, explicit CoT and No-CoT from Dilgren & Wiegreffe (2026), whose reported accuracies we reproduce within half a point. Our models run at

20 Euler steps and guidance 4, with one fixed noise draw per question so that every probe reads the same thought.

E.1 USEFUL: THE TWIN TEST

Table 15: Twin test on 824 pairs. Share of pairs (%) whose answer follows the injected thought (the twin's answer), keeps the original answer, or is neither. Clean follow: the share that follows, counted only on the pairs where the model answers both the question and its twin correctly without any injection.

Model	Follows	Keeps original	Neither	Clean follow
Explicit CoT (reference steps)	97.2	0.2	2.5	99.9
CODI	31.8	19.8	48.4	68.4
PCCoT	27.7	20.8	51.6	62.3
Coconut	7.3	29.5	63.2	19.2
FLaRe, stage 1	**39.7**	**8.7**	51.6	**91.4**
FLaRe, stage 2	37.3	15.0	47.7	76.1

Probe. The twin test asks whether the answer comes from the thought or from the question alone. Each test question gets a twin that changes one number, and we obtain the twin's CoT and answer by recomputing the reference CoT with the new number. For example, the twin of "If Ann is 9 years old and her brother is twice her age, how old will her brother be in 3 years?" makes Ann 8, so the CoT `<<9*2=18>> <<18+3=21>>` becomes `<<8*2=16>> <<16+3=19>>` and the answer changes from 21 to 19. We first run the model on the twin and keep its thought, then run it on the original question with this thought in place of its own. The question now leads to 21 and the thought to 19: the answer follows the thought if it is 19, keeps the question if it is 21, and is neither otherwise. The injected thought is the twin's recomputed CoT for explicit CoT, and for ours the thought that the flow generates for the twin from the same noise draw. We only change a number that appears once in the question and enters a single step, and keep a twin only if all step results remain whole numbers and its new number and answer do not appear in the original CoT, which gives 824 pairs. A model can only follow a thought that leads to the twin's answer, so we also report the clean follow, the follow rate on the pairs where the model, without any injection, answers both the question and its twin correctly. There the injected thought is known to be right, and a model that uses its thought should almost always follow it.

Results. Results are shown in Tab. 15. Explicit CoT follows the injected steps on 97% of the pairs, as a model that reasons through its steps should. CODI follows the injected thought on 32%, PCCoT on 28% and Coconut on 7%, and even on the clean pairs CODI and PCCoT follow on about two thirds and Coconut on one fifth. Our stage 1 and stage 2 models follow on 40% and 37% of all pairs, and on 91% and 76% of the clean pairs. When they do not follow, they rarely keep the original answer and mostly give neither, because the twin's thought is itself wrong: on 98% and 96% of the pairs with neither answer, our models also get the twin wrong without any injection. Replacing the thought altogether shows the same dependence. The stage 1 model falls from 55.3 to 47.4 with the mean thought, to 40.6 with no thought and to 5.9 with the thought of another question. Stage 2 falls from 59.1 to 47.3, 39.3 and 21.6, relying a little more on the question after self-training.

E.2 DIVERSE: SAMPLING BY NOISE

Probe. We draw 16 samples per question. For ours, these use 16 noise draws of the flow, with the first serving as the greedy baseline. The baselines map a question to a single thought, so we add Gaussian noise to it, with a norm of σ times that of the thought, at the largest σ in $\{0.03, 0.1, 0.3, 0.5, 1, 2\}$ that keeps the single-sample accuracy within two points of greedy on the 500 validation questions. A committed error is a greedy error that every sample repeats.

Results. Results are shown in Tab. 16 and Fig. 6. Noise changes the answer of the baselines on a fifth to a half of the questions, yet their pass@16 is at most 5 points above greedy, voting never beats greedy, and half of the errors of CODI and PCCoT are committed. On the decoded reading, our stage 1 model reaches a pass@16 of 77.8 against a greedy 61.2, with 3.4 distinct answers per question, a 3.3-point gain from voting and only 6% of greedy errors repeated by every sample, and

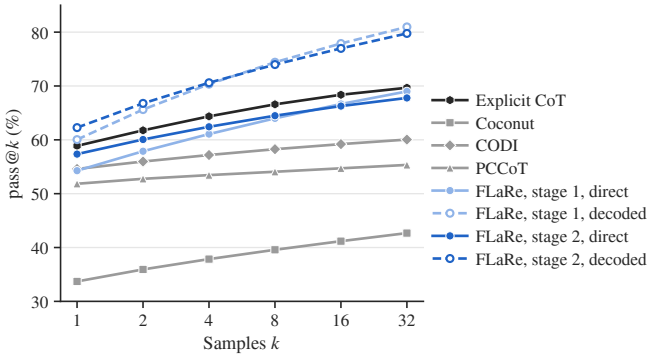


Figure 6: pass@k on the GSM8K test set, estimated from 32 samples per question with the unbiased estimator of Chen et al. (2021). Our models generate flow samples from noise, the baselines add noise to their thought at the calibrated scale, and explicit CoT samples its tokens at temperature 0.7.

Table 16: Sampling $K = 16$ thoughts per question: accuracies (%), distinct answers per question, and committed errors as a share of the greedy errors.

Model	σ	Greedy	pass@1	pass@K	Majority	Distinct ans.	Committed err.
CODI	0.5	55.6	54.6	59.1	54.9	1.46	51%
PCCoT	1.0	54.1	51.9	54.9	52.2	1.31	58%
Coconut	2.0	36.0	33.8	40.6	34.5	1.89	34%
FLaRe, stage 1, decoded	–	61.2	60.1	**77.8**	**64.4**	**3.41**	**6%**
FLaRe, stage 1, direct	–	55.3	–	67.1	55.6	2.68	–
FLaRe, stage 2, decoded	–	**62.6**	**62.0**	77.3	**64.4**	2.89	9%
FLaRe, stage 2, direct	–	59.1	–	66.2	58.2	1.93	–

stage 2 keeps most of this coverage. The gap grows with more samples: from one to 32 samples, stage 2 gains 17.5 points, the baselines 3.5 to 9.0 and explicit CoT 10.8.

E.3 Explainable: Reading the Thoughts

Table 17: Values in order: questions (%) whose reference intermediate results all appear in order, through the LM head for the baselines and in the decoded CoT for ours. Random: the same test with random values.

Model	Values in order	Random
CODI	38.5	0.1
Coconut	19.1	0.2
PCCoT	7.8	0.2
No-CoT, 6 pauses	3.8	0.3
FLaRe, stage 1	46.0	0.1
FLaRe, stage 2	**48.6**	0.1

Table 18: Faithfulness (%) with the twin's thought injected on the pairs whose edit changes an intermediate value. Reads: the decoded thought shows the twin's values. Follows: the answer is the twin's. Reads and follows: both hold, so the answer matches what the thought shows.

Model	Reads twin values	Follows twin	Reads and follows
CODI	24.8	40.3	16.1
PCCoT	32.7	34.9	13.5
Coconut	44.7	8.9	6.4
FLaRe, stage 1	81.0	**44.1**	40.5
FLaRe, stage 2	**81.5**	**44.1**	**41.8**

Probe. We decode each thought and check whether every intermediate result of the reference CoT appears in it in order, which we call values in order, with random values of the same length as the chance level. For ours, the frozen decoder turns the thought into a symbolic CoT, which must contain each result as an exact value, the final one included. The baselines have no decoder, so we read each thought through the model's own LM head and keep its top-1 token. A result counts as read when its first token matches one of these tokens, since Llama splits numbers above 999 into several tokens. A No-CoT model read at six inserted pause positions gives the level reached without any thought. For faithfulness, on the twin pairs whose edit changes an intermediate value, we read

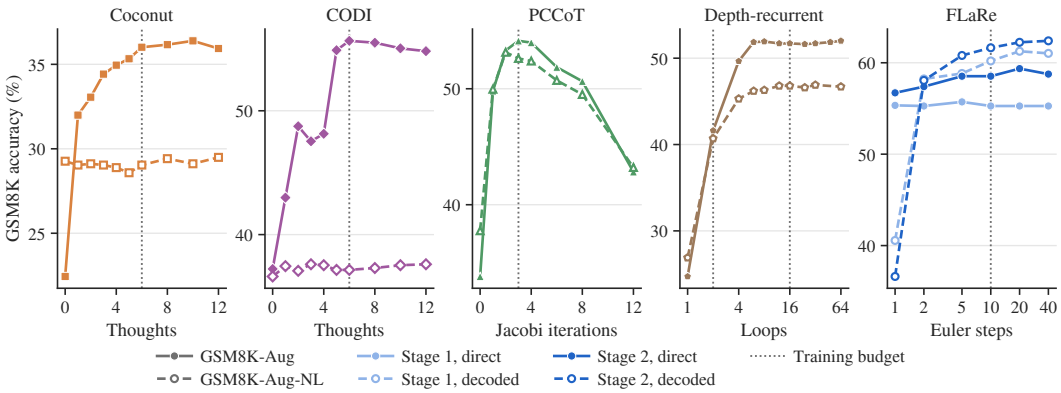


Figure 7: Accuracy against the latent computation spent at inference: thoughts for Coconut and CODI, Jacobi iterations for PCCoT, loops for the depth-recurrent model, and Euler steps for our flow model, on GSM8K-Aug and GSM8K-Aug-NL. The dotted lines mark the training budget.

the injected thought in the same way, decoded for ours and through the LM head for the baselines, and check whether it shows the twin's values and whether the answer follows them.

Results. Results are shown in Tab. 17 and Tab. 18. The values are decoded in order for 46% and 49% of the questions with our stage 1 and stage 2 models, against 39% for CODI, 19% for Coconut and 8% for PCCoT, although our match is stricter and counts a correct CoT that takes another route as a miss. All models stay above the No-CoT control (3.8%) and far above chance (at most 0.3%), so these readings do come from the thoughts. On the twin pairs, our decoded thought shows the twin's values on 81% of the pairs, against 25 to 45% for the baselines, and it both shows them and is followed by the answer on 41% and 42%, against 6 to 16%. When our answer follows the twin, the decoded thought shows it in over 90% of the cases, against about 40% for CODI and PCCoT, whose answers mostly follow the twin without the reading showing it. Coconut shows the opposite pattern, its thought showing the twin's values on 45% of the pairs while its answer follows the twin on only 9%. The alignment of our models remains partial, since on about 40% of the pairs the decoded thought shows a twin value but the answer is not the twin's, mostly because the twin's thought goes wrong after that value and the answer lands on neither.

E.4 Refinable: Accuracy against Inference Budget

Probe. We run each frozen model below and beyond its training budget: 0 to 12 thoughts for Coconut and CODI (trained with 6), 0 to 12 Jacobi iterations for PCCoT (trained with 3), and 1 to 40 Euler steps for ours (trained with 10 in stage 2). The refinable column of Tab. 7 puts all methods on the compute scale of explicit CoT, 24 sequential passes, and reports the gain from about a tenth to half of it: 2 to 12 thoughts or iterations for the baselines, and 1 to 5 Euler steps for ours, that is 2 to 10 passes with the two guidance branches. Fig. 7 also shows each baseline trained on GSM8K-Aug-NL, which writes the same steps as sentences, and a depth-recurrent model (Geiping et al., 2025; McLeish et al., 2025) that we train ourselves, looping the middle layers of the same base model 2 to 16 times.

Results. Results are shown in Fig. 7. Coconut stays flat, CODI loses about a point at twice its budget, PCCoT drops from 54.1 at three iterations to 42.8 at twelve, and the depth-recurrent model is flat from six loops to 64. On the decoded reading, our stage 1 model rises from 40.6 at one step to 58.2 at two and 61.3 at twenty, while its direct reading is 55.3 from the first step on. Stage 2, whose answer pass is trained on the 10-step rollout, gains on both readings from one to twenty steps, from 36.6 to 62.6 decoded and from 56.7 to 59.1 direct. Neither model loses more than a point at forty steps.

E.5 Efficient: Latency Measurement

Probe. We time each method from the question to its last answer token, one question at a time on one RTX 3090 in bf16, and report the median over 100 test questions. Each baseline runs its

Table 19: Cost per GSM8K question: accuracy (%), sequential passes before the first answer token, generated tokens and median latency (ms) over 100 questions. FLaRe: stage 2, direct at S Euler steps, decoded at 20.

Method	Acc.	Passes	Tokens	Latency
No-CoT	30.3	0	3.2	40
Explicit CoT	59.3	24	27.6	267
Coconut, 6 thoughts	36.0	6	3.2	104
CODI, 6 thoughts	55.6	6	6.2	136
PCCoT, 3 iterations	54.1	3	1.2	69
FLaRe, direct, $S = 1$	56.7	2	2.2	**47**
FLaRe, direct, $S = 2$	57.4	4	2.2	68
FLaRe, direct, $S = 5$	58.5	10	2.3	133
FLaRe, direct, $S = 10$	58.5	20	2.3	242
FLaRe, direct, $S = 20$	59.1	40	2.2	461
FLaRe, decoded, $S = 20$	**62.6**	40 + 26	25.8	959

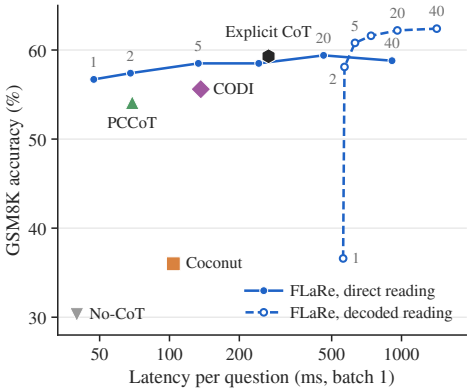


Figure 8: Accuracy against latency per question. Our curves vary the number of Euler steps, shown beside each point.

reference inference code, and ours runs the stage 2 model with S Euler steps of two forward passes each, followed by the direct reading or by the decoded reading, which adds greedy decoding by the frozen 3B decoder.

Results. Results are shown in Tab. 19 and Fig. 8. One Euler step costs 22 ms and one token of explicit CoT 9.7 ms, so twenty steps cost as much as a CoT of about 45 tokens. Our direct curve lies above and to the left of CODI and PCCoT from the first step, at 56.7 in 47 ms, and flattens from five steps, where it is within a point of explicit CoT at half its latency. At two steps, it reaches 57.4 in 68 ms, at half the latency of CODI and a quarter of that of explicit CoT, and at twenty steps it nearly matches explicit CoT at 1.7 times its latency. The decoded reading serves only as a probe and is never used to answer at inference. Its curve is shifted right by half a second of decoding, more than the reasoning itself. It starts at 36.6 after a single step and rises to 60.8 at five and 62.6 at twenty, above every other point from five steps on. Since the decoder reads the thought alone, without the question, this rise shows that the extra steps refine the thought itself: after one step it is a rough estimate that often decodes into a wrong CoT, and each further step brings it closer to a correct one. The direct reading, in contrast, gains less than three points after its first step, since stage 1 trains the answer pass on the model's own one-step endpoints, so it can read the answer from a rough thought.

F Extended Related Work

Tab. 20 first places each method on the design axes of the paper: the family, where the thought lives, how it is produced, what supervises it, whether it can be decoded and how, and whether the inference budget is fixed at training or adjustable. Earlier methods keep the thought in the model's embeddings or hidden states, produce it one slot at a time or through a fixed number of iterations, learn it from an explicit CoT teacher, and read it, if at all, through the LM head at a budget fixed at training. The diffusion family and ours generate the thought in a learned latent space by denoising, with an adjustable budget, and decode it through the VAE decoder.

Latent reasoning. Latent reasoning methods (Chen et al., 2025; Zhu et al., 2025b) move the intermediate computation of explicit CoT (Wei et al., 2022) from discrete tokens into an implicit hidden computation, conducted either vertically over a fixed set of input slots, within the hidden states (Deng et al., 2023) or by looping (Saunshi et al., 2025), or horizontally with added tokens or blocks of tokens, without generating an intermediate text sequence at inference.

Implicit CoT. The first family transfers the thinking from explicit next tokens into the hidden states (Deng et al., 2023; 2024; Yu et al., 2024), as models already perform some multi-hop reasoning latently (Yang et al., 2024b; Biran et al., 2024; Lindsey et al., 2025).

Filler tokens. Pause or filler tokens appended between the input and the answer (Goyal et al., 2024; Pfau et al., 2024), or after each word in recurrent language models (Herel & Mikolov, 2024), widen

Table 20: Latent reasoning methods along the design axes of the paper.

Method	Family	Thought lives in	Produced by	Trained with	Decodable	Budget
Explicit CoT	Horizontal	Tokens	AR decoding	CoT SFT	Written out	Model-chosen
iCoT	Vertical	Hidden states	Single pass	CoT curriculum	No	Fixed
Pause tokens	Vertical	Hidden states	Single pass	Answer loss	No	Fixed
Coconut	Horizontal	Embeddings	AR feedback	CoT curriculum	LM head	Fixed
CODI	Horizontal	Embeddings	AR feedback	CoT distillation	LM head	Fixed
PCCoT	Parallel	Embeddings	Jacobi iteration	CoT distillation	LM head	Fixed
KaVa	Parallel	KV cache	Jacobi iteration	KV distillation	Not reported	Fixed
Looped models	Vertical	Hidden states	Looping	LM loss, no CoT	No	Adjustable
LaDiR	Parallel	Latent space	Denoising	Diffusion on codes	VAE decoder	Adjustable
FLaRe	**Parallel**	**Latent space**	**Denoising**	**Flow matching, self-training**	**VAE decoder**	**Adjustable**

the parallel computation before the answer without adding sequential feedback steps, akin to register tokens in vision Transformers (Darcet et al., 2024).

Continuous feedback. Autoregressive feedback methods, popularized by Coconut (Hao et al., 2025), keep the mechanism of explicit CoT but feed the continuous representation of each step back as the next input instead of a discrete token, as in CODI (Shen et al., 2025) among others (Cheng & Van Durme, 2024; Zhang et al., 2025b; Tan et al., 2025; Wei et al., 2026). These methods are analyzed theoretically by Zhu et al. (2025a).

Looped computation. Looped or recurrent methods (Dehghani et al., 2019; Giannou et al., 2023; Fan et al., 2025; Saunshi et al., 2025; Geiping et al., 2025; Zhu et al., 2025c) keep the number of slots fixed and apply the same layers over them several times, so the reasoning becomes an iterative computation within the hidden states (Wang et al., 2025; Jolicoeur-Martineau, 2025; Baek et al., 2026). Newer methods loop over a fixed block of slots in parallel instead of one step at a time, as in PCCoT (Wu et al., 2025) and KaVa (Kuzina et al., 2026).

Whatever their use of computation, most of these methods still use the explicit CoT in training, through a curriculum as in Coconut and stepwise internalization (Deng et al., 2024), or distillation (Deng et al., 2023) as in CODI and KaVa.

Diffusion. Latent diffusion models for text (Lovelace et al., 2023; Zhang et al., 2023; Meshchaninov et al., 2025) generate by denoising a latent variable from Gaussian noise. In reasoning and planning, the latent is denoised, conditioned on the input, into a state that encodes a step or the full trajectory, followed by autoregressive generation, as in LaDiR (Kang et al., 2026) and Lovelace et al. (2025), with the latent space induced by a VAE trained on explicit CoT (Kang et al., 2026).