FLOW REASONING MODELS: TURNING FLOWS INTO EFFICIENT RECURRENT REASONERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Structured reasoning requires making and revising interdependent decisions to reach a globally consistent solution. Existing architectures struggle with this: autoregressive models cannot revise emitted tokens, while masked diffusion models require careful decoding schemes to coordinate interdependent predictions. We introduce Flow Reasoning Models (FRMs), a novel framework for structured reasoning that adapts continuous flows over discrete structured outputs with a simple recurrent refinement mechanism. By self-conditioning a flow model on its own past outputs, we turn one-shot denoising into iterative solution refinement. This lets FRMs make and revise decisions in parallel, so interdependent decisions are updated jointly. Yet conventional self-conditioning becomes unreliable at greater recurrent depth due to exposure bias between one-step training predictions and recursively generated inference states. We address this mismatch with Fixed-Point Forcing (FPF), which trains FRMs on states produced by their own inference dynamics while preserving the standard flow-matching objective. FRMs achieve solve rates of 99.9%, 99.98%, and 100.0% on Sudoku-Extreme, Zebra, and Maze-Unique, respectively. On Sudoku-Extreme, FRMs achieve higher solve rates than the evaluated masked-diffusion and specialized reasoning baselines while remaining highly compute-efficient, matching EqR's 99.79% full-test solve rate with approximately $51\times$ fewer inference FLOPs.

1 INTRODUCTION

Structured reasoning is difficult because local decisions must be coordinated to satisfy global constraints. Autoregressive decoders commit to tokens in a fixed order and cannot revise a token once it is emitted, even when its global consequences later become clear. Masked diffusion models (Sahoo et al., 2024) offer a flexible generation order and predict multiple tokens in parallel, but tokens updated within the same step are conditionally independent given the current state and cannot condition on one another's realized values, making aggressive parallel updates prone to inconsistencies (Kim et al., 2025). Tightly constrained problems thus require many conservative updates, and correcting earlier errors requires mechanisms such as remasking (Wang et al., 2025b).

Continuous flow models over discrete data offer a promising alternative, jointly evolving continuous states for all output positions throughout sampling. Recent work has demonstrated that continuous flow models can effectively perform unconditional generation of natural language (Lee et al., 2026; Hu et al., 2026), but their ability to perform structured reasoning remains largely unexplored. Indeed, in our own experiments, a naive application of such continuous flows to structured reasoning tasks like Sudoku solves only approximately 13% on the Sudoku-Extreme benchmark (Wang et al., 2025a).

We introduce Flow Reasoning Models (FRMs), which turn continuous flows over discrete outputs into recurrent reasoning models that iteratively refine candidate solutions. Self-conditioning (Chen et al., 2023) feeds a model's previous prediction back as an extra input and has since been adopted for language modeling (Chen et al., 2026). FRMs feed each prediction back alongside the evolving flow state, decomposing complex reasoning into successive refinement steps. This yields an interpretable recurrent state that can be directly decoded into the model's current candidate solution. It also provides a training-efficient form of recurrence that requires no backpropagation through time and retains the canonical flow-matching objective. Self-conditioning alone raises the Sudoku-Extreme solve rate from 13% to 33% (Table 2b), yet most puzzles remain unsolved.

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

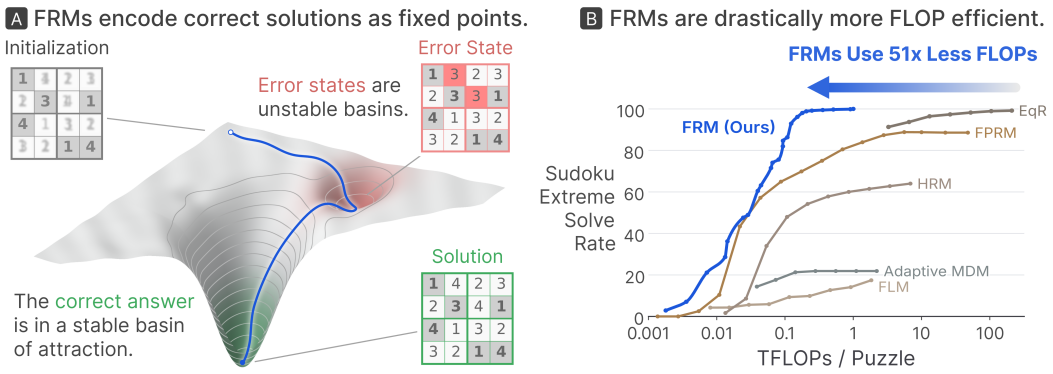


Figure 1: **Flow Reasoning Models turn stable recurrent dynamics into efficient reasoning.** Left: a conceptual view of the recurrent solution landscape. Error states occupy unstable basins, while the correct solution forms a stable attractor that recurrent sampling converges toward. Right: solve-rate–compute curves on EqR's standard Sudoku-Lite subset. The $51\times$ annotation reports the matched 99.79% operating points on the complete Sudoku-Extreme test set, where FRM uses fewer FLOPs than EqR. The landscape is an explanatory schematic rather than a literal three-dimensional embedding of model states.

This limitation becomes clearer through a dynamical-systems interpretation of FRMs. Each recurrent rollout traces a trajectory over candidate solutions, and useful computation should draw that trajectory toward the correct one. In this view, correct solutions should behave as stable, error-correcting attractors, while incorrect states remain transient. Conventional self-conditioning, however, does not reliably produce these dynamics: it can instead converge to confidently incorrect fixed points. We trace this failure to exposure bias: a gap between the self-conditioning states encountered during training and inference. During training, the conditioning state comes from a single model prediction of a partially noised ground-truth solution. At inference, the model begins from noise and repeatedly feeds its own predictions back as conditioning (Bengio et al., 2015; Huang et al., 2025).

We introduce Fixed-Point Forcing (FPF), which trains FRMs to refine the same kinds of predictions they produce during inference. FPF first runs the model recurrently for several steps, then uses its intermediate predictions as the self-conditioning input during training, while leaving the standard flow-matching objective unchanged. This closes the training–inference gap and produces the desired attractor structure: recurrent refinement moves predictions toward the ground truth, correct solutions converge while incorrect states remain dynamically active, and convergence becomes strongly predictive of correctness.

Our core contributions are as follows:

1. **Flow Reasoning Models (FRMs), a novel generative modeling framework that turns continuous flows over discrete structured outputs into recurrent models for reasoning.** Their predictions form an interpretable, directly decodable reasoning state while the models retain the canonical flow-matching objective and require no backpropagation through time.
2. **Fixed-Point Forcing (FPF), a novel training method for recurrent self-conditioned flow models.** We discover that exposure bias can cause these models to converge to spurious fixed points. FPF mitigates this mismatch by training on rollout-derived conditioning states, making additional recurrent depth productive and convergence predictive of correctness.
3. **Residual-guided adaptive inference from learned convergence dynamics.** Because FPF aligns convergence with correctness, the fixed-point residual can guide computation: stop converged trajectories early and restart non-convergent ones from new noise, reducing cost while recovering problems missed by a single trajectory (Sudoku-Lite: $10\times$ lower cost from stopping; 99.9% with restarts).
4. **Near-perfect accuracy and favorable efficiency across structured reasoning tasks.** Across Sudoku-Extreme, Maze-Unique, and Zebra, FRMs reach solve rates of 99.9–100% and favorable accuracy–compute tradeoffs against a variety of methods. On Sudoku-Extreme, FRMs reach a solve rate of 99.9% and match EqR's 99.79% full-test solve rate (Huang et al., 2026) with approximately $51\times$ fewer inference FLOPs (Figure 1).



Figure 2: **Recurrent refinement progressively solves a Sudoku-Extreme instance.** Starting from the fixed clues (gray), we decode the FRM's prediction after each recurrent attempt, with incorrect entries shown in red. The rollout illustrates how recurrent computation revises globally inconsistent early predictions rather than committing to them permanently.

2 Flow Reasoning Models

Flow Reasoning Models cast conditional reasoning as iterative refinement over discrete solutions. We consider data pairs $(c, y) \sim p_{\text{data}}$, where c specifies the observed problem and $y = (y_1, \ldots, y_L)$ is its discrete solution over vocabulary $\mathcal{V}$. FRMs generate only the solution, while c remains fixed as a conditioning input throughout training and inference. We first establish the conditional flow formulation and notation used throughout, then show how self-conditioning equips this model with an explicit recurrent state for reasoning.

2.1 Preliminaries: Discrete Flows for Conditional Reasoning

Following work that models discrete data through continuous or simplex-valued representations (Austin et al., 2021; Gat et al., 2024; Chen et al., 2023; Stark et al., 2024), we encode y as a one-hot endpoint $x_1 \in \{0, 1\}^{L \times |\mathcal{V}|}$ and decode predictions by positionwise argmax.

Flow matching connects Gaussian noise $\varepsilon \sim \mathcal{N}(0, I)$ to this endpoint along a probability path (Lipman et al., 2023; Albergo et al., 2025). We use the linear interpolant

$$x_t = (1 - t)\varepsilon + t x_1, \qquad t \in [0, 1]. \tag{1}$$

We adopt the categorical clean-prediction parameterization of Flow Language Models (Lee et al., 2026): a denoiser $D_\theta^t(x_t \mid c)$ predicts a categorical distribution over the clean token at each position. The corresponding clean endpoint prediction determines the probability-flow velocity:

$$v_\theta^t(x_t \mid c) = \big(D_\theta^t(x_t \mid c) - x_t\big)/(1 - t). \tag{2}$$

The same output is therefore both a directly decodable candidate solution and the quantity used to advance the flow. We train it with tokenwise cross-entropy,

$$\mathcal{L}_{\text{CE}}(\theta) = \mathbb{E}_{t,c,y,\varepsilon}\Big[-\sum_{i=1}^{L} \log D_\theta^t(x_t \mid c)_{i,y_i}\Big]. \tag{3}$$

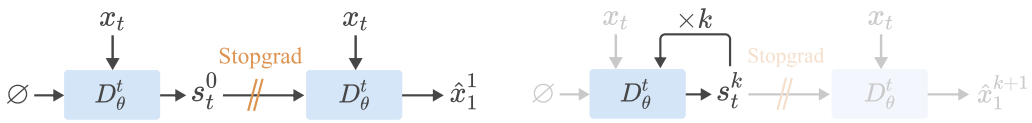
This standard discrete-flow model is our FLM baseline. On its own it leaves a substantial gap on structured reasoning; FRMs add a recurrent axis while retaining this path and objective.

2.2 Self-Conditioning as Recurrent Reasoning

We seek a recurrent model that can repeatedly revise a candidate solution. Self-conditioning provides a natural form of recurrence for flow-based models because the denoiser already predicts the clean endpoint at every evaluation (Chen et al., 2023). FRMs feed this prediction back as an explicit carry state for the next evaluation. The denoiser itself remains a stateless map, and recurrence is externalized through its outputs rather than encoded in an unconstrained hidden state. Consequently, every intermediate carry has a fixed interpretation: it is a complete candidate solution that can be decoded, inspected, and directly supervised. Writing the self-conditioned denoiser as $D_\theta^t(x_t \mid c, s)$, we use $s = \varnothing$ to denote no self-conditioning, implemented as a zero-valued null carry. Recurrent refinement takes the form

$$s_t^{(0)} = \varnothing, \qquad s_t^{(k+1)} = D_\theta^t(x_t \mid c, s_t^{(k)}), \tag{4}$$

Repeated application allows the model to preserve consistent decisions, revise inconsistent ones, and correct earlier mistakes without introducing a separate learned recurrent state.



(a) Conventional self-conditioning (b) Fixed-Point Forcing

Figure 3: **Fixed-Point Forcing replaces one-pass carries with rollout-derived carries.** Conventional self-conditioning forms a detached carry with one denoiser pass at the supervised interpolant. Fixed-Point Forcing instead forms the detached carry through repeated application of the denoiser, exposing the loss-bearing update to a state produced by the model's recurrent inference dynamics.

Conventional self-conditioning training. Self-conditioning is conventionally trained using two denoiser evaluations. The first produces a preliminary clean prediction, which is supplied as a fixed input to the second:

$$\widetilde{s} = D_\theta^t(x_t \mid c, \varnothing), \qquad \widehat{x}_1 = D_\theta^t(x_t \mid c, \widetilde{s}). \tag{5}$$

The standard flow-matching loss in Eq. (3) is applied to $\widehat{x}_1$, while $\widetilde{s}$ is treated as fixed. Thus, conventional self-conditioning teaches the model to refine a carry produced by one preliminary pass, without backpropagating through an unrolled recurrent computation. At inference, however, the carry is generated recursively over many successive applications. This train–inference mismatch limits the benefits of additional recurrent depth, motivating the fixed-point analysis and training method developed next.

3 Training Stable Recurrent Flow Dynamics

Self-conditioning turns the denoiser into an iterative system over candidate solutions. Given a problem c and input state x, each application of the denoiser updates the carry using its previous prediction. A useful recurrent denoiser should refine its own predictions so that repeated application converges to a carry representing the correct solution $y^\star$:

$$s^{(k)} \longrightarrow s^\star, \qquad D_\theta(x \mid c, s^\star) = s^\star, \qquad \mathrm{decode}(s^\star) = y^\star. \tag{6}$$

The solution is attracting when nearby carry states also move toward $s^\star$ under continued refinement. A stable carry that instead decodes to an incorrect answer is a *spurious fixed point*. Conventional self-conditioning does not reliably produce the desired dynamics and can reinforce its own errors.

3.1 Exposure Bias in Recurrent Self-Conditioning

Exposure bias arises when the conditioning states supplied during training differ from those induced by the model at inference (Bengio et al., 2015; Huang et al., 2025). Self-conditioned flows inherit this train–inference mismatch.

During conventional training, a ground-truth-derived interpolant $x_t = (1-t)\varepsilon + tx_1$ is passed through the denoiser once to produce a detached carry, and a second pass is trained using that carry (Fig. 4, left). During inference, the target is unavailable: both the carry and the evolving input $\widehat{x}_t$ are instead produced recursively from the model's own predictions. Consequently,

$$p_{\mathrm{train}}(x_t, s \mid c, t) \neq p_{\mathrm{infer}}^{(k)}(\widehat{x}_t, s^{(k)} \mid c, t), \tag{7}$$

and the discrepancy can grow with recurrent depth. Errors in either state are therefore fed back into later predictions. Because conventional training rarely exposes the denoiser to these self-generated states, it may reinforce an error rather than correct it, eventually producing a spurious fixed point.

3.2 Fixed-Point Forcing

Fixed-Point Forcing (FPF) replaces the one-pass carry used in conventional self-conditioning with a detached carry produced by the model's own multi-step inference dynamics. It thereby trains each recurrent update on the kinds of states produced by earlier updates, directly targeting the train–inference mismatch in Eq. (7).

Conventional Self-Conditioning ⟶ Fixed-Point Forcing

```
t = uniform(0, 1)
x_t = (1 - t) * noise + t * target
- pred = denoiser(
-   x_t, t, cond=clues, carry=None
- )
carry = stopgrad(pred)
refined_pred = denoiser(
    x_t, t, cond=clues, carry=carry
)
update(
    cross_entropy(refined_pred, target)
)
```

```
+ # Construct a rollout-derived carry.
+ t_start = uniform(0, t)
+ x_k = (1 - t_start) * noise
+          + t_start * target
+ carry = None
+ for t_k in linspace(t_start, t, K):
+   pred = denoiser(
+     x_k, t_k, cond=clues, carry=carry
+   )
+   carry = pred
+   x_k = sampler_step(x_k, pred, t_k)
```

Figure 4: **Conventional self-conditioning and Fixed-Point Forcing training.** Both procedures supervise canonical interpolants. Conventional self-conditioning obtains its carry from one pass, whereas FPF collects detached carries along a multi-step rollout and uses them to supervise multiple canonical flow states.

The conceptual difference is illustrated in Fig. 3, and the corresponding training-code change is shown in Fig. 4. We construct the FPF conditioning states by sampling a rollout start t_{start} and running self-conditioned Euler integration over a sequence of later times. At each selected supervision position t_j, we detach the entering carry $s_{\text{FPF}}^{(j)}$ and pair it with a canonical interpolant x_{t_j} for the loss-bearing prediction $D_\theta^{t_j}(x_{t_j} \mid c, \text{stopgrad}(s_{\text{FPF}}^{(j)}))$; gradients do not pass through the rollout.

Crucially, this construction preserves the canonical flow-matching path exactly. As in conventional self-conditioning, every loss-bearing input remains $x_{t_j} = (1 - t_j)\varepsilon + t_j x_1$, the target remains y, and the clean-target cross-entropy is unchanged; rollout predictions enter only through the self-conditioning channel. FPF therefore changes the distribution of what the denoiser conditions on, not the states at which the flow objective is supervised. Training on rollout-derived carries reduces the carry-side exposure bias and the resulting overconfidence and miscalibration, while deeper rollout states expose the model to the smaller, subtler residual errors that remain near convergence and train the local corrections needed around a fixed point.

3.3 Residual-Guided Adaptive Inference

FPF introduces no explicit convergence loss. Let $p_k = \text{softmax}(s^{(k)})$ denote the normalized categorical prediction at recurrent step k. To measure whether recurrent inference has stabilized, we define the token-averaged Euclidean distance between adjacent predictions as

$$r_k = |\mathcal{F}|^{-1} \sum_{i \in \mathcal{F}} \|p_{k,i} - p_{k-1,i}\|_2, \tag{8}$$

where $\mathcal{F}$ contains the generated, non-clue positions. We use this residual to allocate inference compute: converged trajectories stop early, while trajectories that exhaust their depth budget without converging are restarted from new noise. A small residual indicates that the recurrent state has stabilized, but does not by itself certify correctness.

4 Experiments

We evaluate whether Flow Reasoning Models (FRMs) provide accurate and compute-efficient iterative reasoning across three structured-prediction benchmarks. Our primary metric is exact solve rate; we compare inference efficiency using total FLOPs per instance and NFE (see Appendix A).

Datasets. We evaluate Sudoku-Extreme for constraint satisfaction (Wang et al., 2025a), Zebra for relational deduction (Shah et al., 2024), and Maze-Unique for path finding (Huang et al., 2026). These benchmarks differ substantially in sequence length, vocabulary, and output structure. For headline Sudoku results, we use the complete official test split (422,786 unique puzzles after deduplication); configuration-matched frontiers use EqR's fixed 2,048-puzzle Sudoku-Lite split (Huang et al., 2026). For the headline Zebra result, we use the held-out test split of Shah et al. (2024). For Maze-Unique, we use EqR's official, revision-pinned 1,000-instance test split (Huang et al., 2026) (see Appendix A.1).

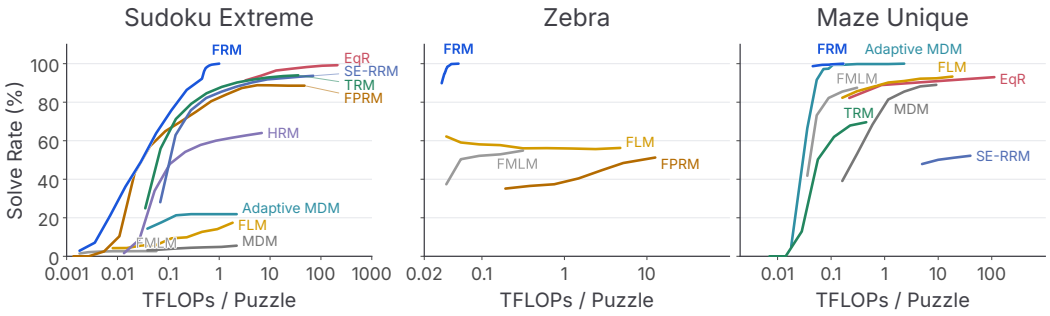


Figure 5: **Accuracy–compute frontiers across structured-reasoning tasks.** Solve rate versus total inference FLOPs per instance on Sudoku-Lite, Zebra, and Maze-Unique under a common FLOP-counting convention. The Sudoku FRM curve scales depth through 512 NFE and then restart breadth through 256; other FRM panels use measured adaptive operating points. The SE-RRM Maze curve evaluates transfer from Maze-Hard to Maze-Unique. The Zebra FPRM curve reports seed 1, matching the tabulated result. Only configuration-matched curves are shown.

4.1 Accuracy and Efficiency Across Structured Reasoning Tasks

Flow Reasoning Models achieve near-perfect exact solving across distinct reasoning domains. FRMs reach solve rates of 99.9% on Sudoku-Extreme, 99.98% on Zebra, and 100.0% on Maze-Unique (Table 1). After 275K training iterations, FPF with residual-guided sequential restarts solves 99.91% of puzzles on the complete official Sudoku-Extreme test set. Near-perfect performance across all three therefore shows that recurrent flow refinement is not tied to a particular task representation or constraint structure. At the operating point matched to EqR's reported 99.79% full-evaluation result (Huang et al., 2026), FRM reaches 99.7909% at 4.11 mean TFLOPs per puzzle versus 210.0 for EqR under our common FLOP accounting, a $51\times$ reduction; the 99.91% operating point uses 8.08 mean TFLOPs.

FRMs solve more problems with less inference compute. Across all three tasks, FRMs achieve favorable accuracy–compute tradeoffs against a variety of methods (Fig. 5). The Sudoku curve uses EqR's fixed 2,048-puzzle Sudoku-Lite split because full-test curves are prohibitively expensive; the matched full-test result above gives the conservative $51\times$ comparison. FRM likewise dominates the measured Maze-Unique tradeoff and reaches 100.0%. On Zebra, it solves all 1,024 plotted examples and $99{,}984/100{,}000$ (99.984%) examples on the full mixed-size test split. The Zebra frontier contains fewer methods because several specialized recurrent baselines lack task-compatible implementations or checkpoints. We therefore report compute comparisons only for reproduced, configuration-matched methods and use Zebra primarily to evaluate whether the gains from FPF transfer to relational reasoning (see Appendix A.3).

4.2 Self-Conditioning Enables Recurrent Refinement

Vanilla discrete flows achieve limited solve rates and do not scale with inference compute. Across all three tasks, the vanilla FLM baselines plateau well below the full FRM (Table 2b), and allocating more forward passes leaves their solve-rate curves essentially flat. Additional integration more accurately traces the learned probability path, but provides no mechanism for repeatedly revising the current solution.

Self-conditioning enables recurrent refinement. Adding self-conditioning improves solve rate over a plain discrete flow on Sudoku-Extreme ($13.1\% \rightarrow 32.6\%$) and Maze-Unique ($77.6\% \rightarrow 96.2\%$; Table 2b). On Zebra, however, self-conditioning lowers the three-seed mean from 62.3% to 36.9%, and both the base and self-conditioned flows exhibit high seed variance. Feeding each prediction back to the model creates a recurrent computation that can preserve correct assignments and revise earlier mistakes, as illustrated by the decoded trajectory in Fig. 2.

Conventional self-conditioning remains unreliable at depth. Despite these gains, its Sudoku-Extreme solve rate plateaus near one-third of puzzles as recurrent depth increases (Fig. 6, middle), motivating a training procedure that makes the recurrent computation reliable over long rollouts.

Method	Sudoku-Extreme		Zebra		Maze-Unique	
	Acc. (%)	Size	Acc. (%)	Size	Acc. (%)	Size
Diffusion language models and flows						
FMLM (Lee et al., 2026)	2.44	7M	74.2	25M	87.5	8.1M
FMLM+ (Agarwal et al., 2026)	7.13	28.6M	–	–	62.0	28.6M
MDLM (Sahoo et al., 2024)	4.84	30M	76.9†	19M	89.0	8M
ReMDM (Wang et al., 2025b)	7.02	30M	–	–	93.5	8M
FLM (Lee et al., 2026)	17.33	7M	67.9	25M	93.3	8M
Adaptive MDLM (Kim et al., 2025)	20.99	30M	98.3†	19M	100.0	8M
Specialized recurrent reasoners						
HRM (Wang et al., 2025a)	55.0	27.3M	–	–	0.3	27.3M
TRM (Jolicoeur-Martineau, 2025)	87.4	7M	–	–	77.9	0.264M
FPRM (Movahedi et al., 2026)	94.2	7M	51.27	6.84M	100.0	7M
SE-RRM (Freinschlag et al., 2026)	98.84	2M	–	–	52.2	2M
EqR (Huang et al., 2026)	99.79	5.03M	–	–	93.0	2.6M
Flow Reasoning Model	**99.91**	7M	**99.98**	25M	**100.0**	8M

Table 1: **Best measured exact-solve accuracy and model size.** Parameter counts correspond to the evaluated configuration. † marks published values that were not reproduced in our evaluation; dashes mark unavailable results or configuration-matched counts.

4.3 Fixed-Point Forcing Enables Reliable Recurrent Reasoning

Fixed-Point Forcing makes recurrent reasoning robust to exposure bias. With conventional self-conditioning, solve rate plateaus as recurrent depth increases while gold-target cross-entropy grows, so additional updates move predictions farther from the correct target without improving exact solve rate (Fig. 6). Training on rollout-derived carries reverses this behavior: under FPF, additional depth continues to correct solutions, raising mean Sudoku-Extreme solve rate across three seeds from $32.6 \pm 3.7\%$ to $99.7 \pm 0.1\%$ and achieving $99.9 \pm 0.2\%$ on Zebra and $99.2 \pm 0.6\%$ on Maze-Unique (Table 2b). Because FPF changes only the recurrent conditioning distribution, these results isolate exposure to model-generated states as the key ingredient for reliable recurrent reasoning.

Fixed-Point Forcing makes convergence predictive of correctness. Under FPF, additional recurrent updates reduce gold-target cross-entropy as solve rate increases, directing convergent trajectories toward correct solutions rather than merely self-consistent predictions (Fig. 6). The residual between adjacent predictive states remains a nearly perfect correctness signal across depth under FPF but degrades under conventional self-conditioning. This correctness signal generalizes across tasks, achieving AUROC 0.998 on Sudoku, 1.000 on Zebra, and 0.989 on Maze-Unique, and is robust to its definition on Sudoku, with AUROC 0.9993 for both Euclidean distance and symmetric KL and 0.9849 for predictive cross-entropy (Appendix Tables 6 and 7). Although convergence is not a formal certificate of correctness, these findings support the attractor picture in Fig. 1: FPF makes correct solutions stable, error-correcting attractors, whereas conventional self-conditioning can converge to stable incorrect fixed points. This is an empirical dynamical interpretation rather than a claim of literal low-dimensional geometry.

Fixed-time sampling makes recurrent refinement effective. Sampler choice strongly affects how effectively FRMs use additional recurrent updates. On Sudoku-Extreme, repeatedly denoising and renoising at a fixed intermediate noise level reaches 94.6% solve rate, compared with 69.2% for annealed SDE and 30.3% for Euler at the same 4,096-step budget (Appendix B.2). Unlike standard flow sampling, the fixed-time procedure does not anneal from noise to the data endpoint; it repeatedly applies the learned denoiser at one intermediate noise level. This shows that full flow integration is unnecessary at inference, although our experiments do not isolate which parts of flow-based training are essential.

4.4 Residual-Guided Adaptive Inference

The residual enables reliable early stopping. Using the residual defined in Section 3.3 as an internal signal of correctness, we calibrate one Euclidean-residual threshold on held-out validation puzzles and stop after the criterion remains satisfied for eight consecutive checks. On Sudoku-Lite,

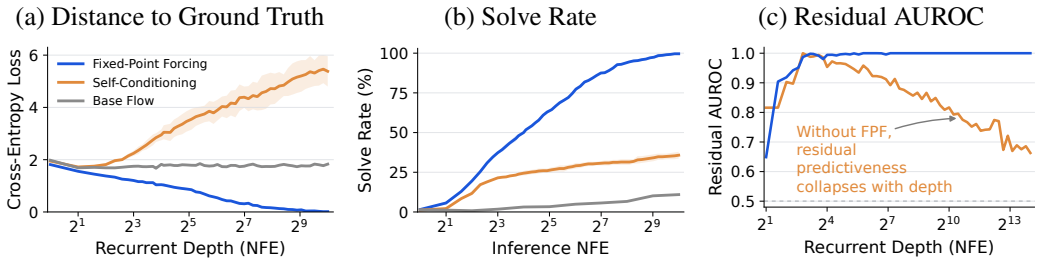


Figure 6: **Fixed-Point Forcing makes recurrence productive and convergence informative on Sudoku-Extreme.** Left: Self-conditioning increases gold-target loss with depth, while FPF drives it toward zero. Middle: FPF solve rate improves with NFE while self-conditioning saturates. Right: The FPF residual predicts correctness across depths, while the self-conditioning residual degrades.

this reduces mean inference cost from 3.67 to 0.36 TFLOPs, a $10.2\times$ reduction, while changing exact solve rate by only 0.24 percentage points (Table 2a). This efficiency gain replicates on Zebra, where the same policy preserves 100% solve rate while reducing mean cost from 4.18 to 0.071 TFLOPs.

Sequential restarts recover the accuracy tradeoff and raise the solve-rate ceiling. Different random initializations can produce different trajectories for the same puzzle. We therefore restart only trajectories that reach their depth limit without satisfying the calibrated criterion, and retain the lowest-residual prediction if the cap is reached. With a maximum breadth of 128 and a maximum depth of 4,096 NFE per attempt, this raises solve rate to 99.90% at 1.02 mean TFLOPs, improving on fixed-depth inference while using $3.6\times$ less computation. The policy uses target solutions only for final scoring, never for stopping, restarting, or candidate selection. The restart gain likewise replicates on Maze-Unique, raising solve rate from 99.5% to 100% at 0.170 mean TFLOPs, $17.1\times$ below matched fixed-depth inference (Appendix B.3).

Depth and breadth provide complementary test-time scaling. Increasing recurrent depth improves refinement within a trajectory, while increasing restart breadth helps escape the smaller set of failures that remain. On Sudoku-Lite, increasing depth from 512 to 4,096 at breadth one reduces failure from 7.81% to 1.66%; at depth 512, increasing breadth from one to 16 reduces it to 0.98% (Appendix C.1). Thus, recurrent depth and stochastic restarts are distinct rather than interchangeable sources of additional test-time computation.

5 Related Work

Diffusion and Flow Language Models. A substantial body of work has developed diffusion-based alternatives to autoregressive modeling for discrete data and language (Austin et al., 2021; Gat et al., 2024; Stark et al., 2024). Masked Diffusion Language Models generate sequences through iterative denoising (Sahoo et al., 2024). Shi et al. (2024) derive a simplified continuous-time objective as a weighted integral of cross-entropy losses and generalize the framework to state-dependent masking schedules. Within this family, Kim et al. (2025) use adaptive token ordering to allocate computation, whereas Wang et al. (2025b) introduce remasking so earlier commitments can be revised; Duo instead uses uniform categorical corruption, keeping every token revisable throughout sampling (Sahoo et al., 2025). More recently, continuous flow language models have emerged as competitive alternatives. FLM learns Gaussian-to-one-hot probability paths and FMLM distills them into few-step maps (Lee et al., 2026), while LangFlow and ELF define flows in learned and contextual embedding spaces, respectively (Chen et al., 2026; Hu et al., 2026). FRMs are inspired by this diffusion-and-flow tradition, but use the clean denoiser prediction as a directly decodable recurrent state and train the resulting closed loop for iterative solution refinement rather than treating repeated evaluations only as steps of a generative sampler.

Recurrent and Fixed-Point Reasoning. Recent structured reasoners scale test-time computation through weight-tied recurrence. HRM uses hierarchical recurrent states (Wang et al., 2025a). TRM (Jolicoeur-Martineau, 2025) and PTRM (Sghaier et al., 2026) repeatedly update compact answer and latent states. EqR explicitly trains solution-aligned attractor dynamics (Huang et al., 2026). FPRM similarly observes that convergence becomes predictive of correctness and uses this

(a) Inference policy.

Policy	Solve (%)	TFLOPs
Fixed-depth FRM	98.34	3.67
+ Early stopping	98.10	0.36
+ Restarts	99.90	1.02

(b) Controlled training ablation.

Recipe	Sudoku-Extreme Acc. (%)	Zebra Acc. (%)	Maze-Unique Acc. (%)
Base Flow	13.1 ± 0.7	62.3 ± 29.1	77.6 ± 16.7
+ Self-cond.	32.6 ± 3.7	36.9 ± 18.0	96.2 ± 3.0
+ FPF	$\mathbf{99.7 \pm 0.1}$	$\mathbf{99.9 \pm 0.2}$	$\mathbf{99.2 \pm 0.6}$

Table 2: **Training and inference ablations.** (a) On Sudoku-Lite, residual-guided early stopping cuts compute, while restarts raise solve rate. (b) FPF is the key training ingredient across tasks; entries are mean $\pm$ sample standard deviation over three seeds under fixed-time, single-trajectory inference without early stopping or restarts.

signal for adaptive halting (Movahedi et al., 2026). FRMs share the emphasis on recurrent test-time scaling, attractor dynamics, and meaningful convergence, but realize them within a self-conditioned flow language model whose clean prediction serves as both recurrent input and explicit solution candidate. FRM retains the standard non-causal Transformer and flow objective already demonstrated at language-model scale (Lee et al., 2026; Hu et al., 2026). FPF also avoids backpropagation through the rollout: model-generated states enter only through the conditioning channel, while every loss-bearing flow state remains on the target-derived canonical path.

Learning from Model-Induced States. Conventional self-conditioning feeds a detached clean-sample estimate into later denoising steps but trains this channel with a one-pass rather than recursively generated prediction, following Analog Bits (Chen et al., 2023). This creates exposure bias: the model is evaluated on its own induced context without being trained on that context (Ross et al., 2011; Bengio et al., 2015). Concurrent work likewise frames self-conditioned flow language models as fixed-point iterations for few-step generation (Yoo et al., 2026); we instead study structured reasoning and training on model-induced recurrent states. Self Forcing addresses the analogous mismatch in autoregressive video diffusion by training on self-generated rollout context (Huang et al., 2025). FPF specializes this idea to the recurrent carry rather than previous frames. It requires neither a sequence-level distribution-matching objective nor differentiation through the rollout, changing only the carry distribution while preserving the canonical flow path and endpoint objective.

6 Discussion

Flow Reasoning Models indicate that discrete flows can be a powerful and efficient model class for structured reasoning. Across constraint satisfaction, relational deduction, and path finding, FRMs achieve strong performance while retaining a favorable accuracy–compute tradeoff. Conditioning discrete flows on their own past predictions creates an iterative refinement process, while training on recursively generated states with Fixed-Point Forcing dramatically improves performance and makes additional recurrent depth productive. Our results also show that the inference procedure is part of the reasoning algorithm rather than an implementation detail: sampler choice substantially changes performance. Fixed-time recurrence and residual-guided stopping and restarts provide complementary ways to allocate compute between deeper refinement and additional attempts without retraining. Together, these results suggest that recurrent reasoning need not rely on a specialized recurrent architecture: it can emerge from the learned refinement dynamics of a general-purpose generative model.

Limitations. Our experiments focus on compact models and structured reasoning tasks; whether the gains persist at larger scales or in less structured domains remains open. Several specialized recurrent baselines lack task-compatible implementations or checkpoints for Zebra, so we use those results primarily to assess transfer to relational reasoning rather than as a comprehensive efficiency comparison. Residual-guided inference treats convergence as a useful signal rather than a correctness certificate, and difficult instances may still require long trajectories or restarts. Finally, FPF adds training computation and implementation complexity relative to conventional self-conditioning.

AI USE STATEMENT

General-purpose language modeling tools were used during implementation, debugging, figure and table preparation, and revision of the manuscript. They were not used as a source of experimental measurements. Results produced in this work were obtained from executed evaluations and checked against saved artifacts; comparison results attributed to prior work were taken from the cited papers. The authors reviewed the generated code and prose, verified the reported results, and take full responsibility for the scientific claims and contents of the submission.

ETHICS STATEMENT

This work studies algorithmic reasoning on synthetic, publicly available structured benchmarks and involves no human participants, personal data, or deployment in high-stakes settings. We are not aware of specific ethical risks arising from the experiments beyond the general responsibilities of accurate reporting, reproducibility, and responsible downstream use of machine-learning methods.

REPRODUCIBILITY STATEMENT

Sections 2–4 specify the model, training objective, inference procedures, evaluation metrics, and principal experimental protocol. The appendix documents dataset construction and fixed evaluation subsets (Appendix A.1), architecture and training hyperparameters (Appendix A.2), baseline implementations (Appendix A.3), compute accounting (Appendix A.4), and the controlled accuracy, ablation, adaptive-inference, and dynamics analyses. We will release the code needed to reproduce the reported tables and figures upon publication.

REFERENCES

Manan Agarwal, Sheel Shah, Chanhyuk Lee, Jaehoon Yoo, Jerry Huang, Seunghoon Hong, Aditi Raghunathan, Jinwoo Kim, and Nicholas M. Boffi. Posterior refinement: Fast language generation via any-order flow maps. *arXiv preprint arXiv:2606.24773*, 2026.

Michael Albergo, Nicholas M. Boffi, and Eric Vanden-Eijnden. Stochastic interpolants: A unifying framework for flows and diffusions. *Journal of Machine Learning Research*, 26(209):1–80, 2025. URL http://jmlr.org/papers/v26/23-1605.html.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 17981–17993. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/958c530554f78bcd8e97125b70e6973d-Paper.pdf.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/e995f98d56967d946471af29d7bf99f1-Paper.pdf.

Ting Chen, Ruixiang Zhang, and Geoffrey Hinton. Analog bits: Generating discrete data using diffusion models with self-conditioning, 2023. URL https://arxiv.org/abs/2208.04202.

Yuxin Chen, Chumeng Liang, Hangke Sui, Ruihan Guo, Chaoran Cheng, Jiaxuan You, and Ge Liu. Langflow: Continuous diffusion rivals discrete in language modeling, 2026. URL https://arxiv.org/abs/2604.11748.

Richard Freinschlag, Timo Bertram, Erich Kobler, Andreas Mayr, and Günter Klambauer. Symbol-equivariant recurrent reasoning models. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR, 2026. URL https://arxiv.org/abs/2603.02193.

Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky T. Q. Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete flow matching. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 133345–133385. Curran Associates, Inc., 2024. doi: 10.52202/079017-4239. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/f0d629a734b56a642701bba7bc8bb3ed-Paper-Conference.pdf.

Keya Hu, Linlu Qiu, Yiyang Lu, Hanhong Zhao, Tianhong Li, Yoon Kim, Jacob Andreas, and Kaiming He. Elf: Embedded language flows, 2026. URL https://arxiv.org/abs/2605.10938.

Benhao Huang, Zhengyang Geng, and Zico Kolter. Equilibrium reasoners: Learning attractors enables scalable reasoning, 2026. URL https://arxiv.org/abs/2605.21488.

Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self forcing: Bridging the train-test gap in autoregressive video diffusion. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 167283–167308. Curran Associates, Inc., 2025. doi: 10.52202/085713-5576. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/f4823f831af67a3ef15e41a85434422a-Paper-Conference.pdf.

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

Jaeyeon Kim, Kulin Shah, Vasilis Kontonis, Sham Kakade, and Sitan Chen. Train for the worst, plan for the best: Understanding token ordering in masked diffusions, 2025. URL https://arxiv.org/abs/2502.06768.

Chanhyuk Lee, Jaehoon Yoo, Manan Agarwal, Sheel Shah, Jerry Huang, Aditi Raghunathan, Seunghoon Hong, Nicholas M. Boffi, and Jinwoo Kim. Flow map language models: One-step language modeling via continuous denoising, 2026. URL https://arxiv.org/abs/2602.16813.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t.

Sajad Movahedi, Vera Milovanović, Shlomo Libo Feigin, Alexander Theus, Thomas Hofmann, Valentina Boeva, T. Konstantin Rusch, and Antonio Orvieto. Fixed-point reasoners: Stable and adaptive deep looped transformers, 2026. URL https://arxiv.org/abs/2606.18206.

Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning, 2011. URL https://arxiv.org/abs/1011.0686.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 130136–130184. Curran Associates, Inc., 2024. doi: 10.52202/079017-4135. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/eb0b13cc515724ab8015bc978fdde0ad-Paper-Conference.pdf.

Subham Sekhar Sahoo, Justin Deschenaux, Aaron Gokaslan, Guanghan Wang, Justin T Chiu, and Volodymyr Kuleshov. The diffusion duality. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=9P9Y8FOSOk.

Amin Sghaier, Ali Parviz, and Alexia Jolicoeur-Martineau. Probabilistic tiny recursive model, 2026. URL https://arxiv.org/abs/2605.19943.

Kulin Shah, Nishanth Dikkala, Xin Wang, and Rina Panigrahy. Causal language modeling can elicit search and reasoning capabilities on logic puzzles. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 56674–56702. Curran Associates, Inc., 2024. doi: 10.52202/079017-1805. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/67b31ca159553d5593e62d7b998d63ea-Paper-Conference.pdf.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 103131–103167. Curran Associates, Inc., 2024. doi: 10.52202/079017-3277. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/bad233b9849f019aead5e5cc60cef70f-Paper-Conference.pdf.

Hannes Stark, Bowen Jing, Chenyu Wang, Gabriele Corso, Bonnie Berger, Regina Barzilay, and Tommi Jaakkola. Dirichlet flow matching with applications to DNA sequence design. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 46495–46513. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/stark24b.html.

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

Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling. In *Advances in Neural Information Processing Systems*, volume 38, pp. 147282–147339, 2025b. doi: 10.52202/085713-4927. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/d877ea0dd78bdbe54830670618c1de09-Abstract-Conference.html.

Jaehoon Yoo, Wonjung Kim, Floor Eijkelboom, Chanhyuk Lee, Nicholas M. Boffi, Seunghoon Hong, and Jinwoo Kim. Self-conditioned flow map language models via fixed-point flows, 2026. URL https://arxiv.org/abs/2607.00714.

A Experimental Details

A.1 Datasets

Sudoku-Extreme. Sudoku-Extreme (Wang et al., 2025a) consists of hard 9×9 puzzles, each with a unique solution. The model generates the full grid, with the given clues clamped. For each training seed, we train on a difficulty-balanced subset of 1,000 puzzles from the official training split, augmented with random Sudoku symmetries. We evaluate on three sets: the complete official test split ($422{,}786$ puzzles after removing duplicates) for the headline result, a fixed difficulty-balanced subset of 1,024 test puzzles for the controlled ablations, and EqR's 2,048-puzzle Sudoku-Lite subset (Huang et al., 2026) for the accuracy–compute curves.

Zebra. Zebra puzzles (Shah et al., 2024) ask for the assignment of attribute values to houses that satisfies a list of clues. We use the released data, about 1.5 million training and 0.1 million test puzzles with three to six houses and three to six attributes, and train and test on all 16 size combinations together; Kim et al. (2025) also evaluate masked diffusion models on this benchmark. We keep each puzzle's clue serialization and replace its solver-ordered answer trace with a fixed attribute-by-house answer grid, padded to 6×6 (sequence length 383). We report exact-grid solve rate on the test split.

Maze-Unique. Maze-Unique (Huang et al., 2026) has official splits of 1,000 training and 1,000 test 30×30 mazes; we use a fixed revision of the released data. Walls, start and goal are clamped, and the model generates the rest of the grid. We use it instead of the more common Maze-Hard because every maze has a unique solution path, so exact-grid accuracy equals path validity and optimality and never penalizes an alternative valid route.

A.2 Models and Training

All models are non-causal DiT denoisers trained with the categorical clean-prediction objective; Table 3 lists their settings. Stage A trains conventional self-conditioning from scratch. Stage B starts from the final Stage-A weights with a fresh optimizer and EMA and trains with FPF. The headline Sudoku-Extreme model continues the seed-0 Stage-B weights at the single time $t = 0.4$ used at

inference, with re-noised FPF rollouts on every batch row and a learning rate of 10^{-4} (Stage C), to 275K training steps in total; it is evaluated with residual-guided restarts (Appendix B). Checkpoints are selected on fixed subsets of the official test partitions: 256 Sudoku-Extreme puzzles, 256 Zebra puzzles, and 128 Maze-Unique instances. Final results are evaluated on the complete reported test sets. For Table 2b, each seed is evaluated with one trajectory, without early stopping or restarts, on 1,024 fixed puzzles for Sudoku-Extreme and Zebra and all 1,000 Maze-Unique test instances. The FPF rows use fixed times 0.4, 0.2, and 0.15 with 8,192, 1,025, and 4,097 NFE, respectively.

Setting	**Sudoku-Extreme**	**Zebra**	**Maze-Unique**
Parameters	7.0M	25.0M	8.0M
Hidden size / blocks / heads	256 / 7 / 8	384 / 12 / 12	256 / 8 / 8
Sequence length / vocabulary	81 / 10	383 / 21	900 / 6
Learning rate / weight decay	3×10^{-4} / 0	3×10^{-4} / 0	1×10^{-4} / 1.0
Puzzles per optimizer step	128	128	64
Warmup steps	0	0	500
Steps in Stage A / B	100K / 100K	200K / 200K	300K / 100K
Self-conditioning probability	0.5	0.5	0.5
FPF share of batch / rollout steps K	0.5 / 16	0.5 / 16	0.5 / 16
FPF supervised states N	4	4	4

Table 3: **Architecture and training settings.** All models use AdamW with $\beta = (0.9, 0.999)$, gradient clipping at 1.0, EMA decay 0.9999, dropout 0.1, and a constant learning rate. The headline Sudoku-Extreme model is trained further at $t = 0.4$ (Stage C) to 275K steps in total.

FPF training step. On half of each batch, FPF draws an end time $t_{\text{end}} \sim \mathcal{U}(0, 1)$ and a start time $t_{\text{start}} \sim \mathcal{U}(0, t_{\text{end}})$, starts from the ground-truth interpolant at t_{start}, and runs $K = 16$ self-conditioned Euler steps to t_{end} without gradients. It then draws $N = 4$ of the $K + 1$ rollout positions and, at each, pairs the detached carry entering that position with a freshly noised ground-truth interpolant at the same time; the N cross-entropies are summed. The other half of the batch uses conventional self-conditioning with probability 0.5. The carry is the model's output logits, soft-capped at ± 30 with a $\tanh$ and fed back through a zero-initialized linear embedding. Times are drawn in the reparameterized time of FLM (Lee et al., 2026).

A.3 Baseline Implementations

We use released weights when compatible with the benchmark and otherwise train official or paper-based implementations on the corresponding data. Compute comparisons contain measured runs only and charge every learned-model evaluation; published results without a compatible profiled checkpoint appear only as headline accuracies.

- **FMLM and FLM** (Lee et al., 2026) [code]. We port the official implementation and train task-specific models where matching weights are unavailable. FMLM students are distilled from the corresponding FLM teachers. The reported Zebra entries use fixed 5×5 puzzles.
- **MDLM, ReMDM, and Adaptive MDLM** (Sahoo et al., 2024; Wang et al., 2025b; Kim et al., 2025) [MDLM code; ReMDM code]. We train MDLM checkpoints on the corresponding task, apply ReMDM as the authors' inference-only sampler, and implement Adaptive MDLM from its published procedure. Our fixed-5×5 Zebra reproduction did not yield a reliable result, so the marked Zebra accuracies are the published mixed-size results.
- **HRM, TRM, FPRM, and EqR** (Wang et al., 2025a; Jolicoeur-Martineau, 2025; Movahedi et al., 2026; Huang et al., 2026) [HRM; TRM; FPRM; EqR]. We use official implementations and released checkpoints where available, otherwise training on the matching dataset with the released recipe. The Zebra FPRM frontier uses seven validation-frozen test operating points from seed 1; the same run supplies the result in Tables 1 and 4.
- **SE-RRM** (Freinschlag et al., 2026) [code]. We train the official 2M-parameter recipe. Its Maze-Unique entry is explicitly a transfer diagnostic from our independently trained Maze-Hard checkpoint; all other measured entries use the named task's training split.

- **Posterior Refinement (FMLM+)** (Agarwal et al., 2026). We use the official implementation [code] with dataset and metric adapters. The released 30-clue Sudoku checkpoint obtains 7.13% exact solve on our fixed, difficulty-balanced Sudoku-Extreme subset. For Maze-Unique, our task-trained 28.6M-parameter model uses four refinement rounds and solves $620/1{,}000$ test mazes (62.0%), compared with $280/1{,}000$ (28.0%) for equal-cost full-sequence flow. We report no reliable Zebra result.

A.4 Inference Compute Accounting

Number of function evaluations. We report the realized number of learned-model evaluations required to solve one puzzle. For our Euler, SDE, and held-time samplers, NFE counts every iterative update and the final readout evaluation; for methods with adaptive stopping, branching, or breadth, we count all evaluations actually executed for that instance. NFE exposes the number of sequential model calls and is easy to reproduce, but it is not a sufficient cross-method compute measure: one evaluation can differ substantially in cost across architectures, sequence lengths, model sizes, and internally recurrent networks. The appendix table reports *solved-puzzle mean NFE*: for each puzzle that is solved at some point along its evaluated trajectory, we record the first NFE at which it is solved and average over solved puzzles only. This is an oracle trajectory diagnostic unless correctness can be detected without access to the ground truth; it should therefore not be interpreted as deployable early stopping for every method.

Floating-point operations. We therefore use total FLOPs per puzzle as the primary hardware-independent compute measure. We profile an actual batch-one forward pass with PyTorch's operator-level FLOP counter, disabling fused attention so that its matrix multiplications are visible, and multiply the measured per-evaluation cost by the realized NFE; methods with nonuniform calls are profiled and summed by call type. This quantity should be interpreted as a lower bound on the work required to solve an instance: it counts floating-point arithmetic but not memory traffic, kernel-launch and framework overhead, control flow, communication, or data movement. We therefore present the full accuracy–compute frontiers rather than compressing each method to a single operating-point ratio.

B Additional Accuracy and Ablation Results

This section documents the headline accuracy–compute comparison and training-recipe ablation; Appendix A gives datasets, baseline implementations, and FLOP accounting.

B.1 Accuracy–Compute Evaluation

Each frontier contains measured operating points only. We vary each checkpoint's native compute control and charge every learned-model call. Fixed-depth methods use the requested depth; adaptive methods use realized mean computation. We neither extrapolate unsupported method–dataset pairs nor include published accuracies lacking a compatible profiled checkpoint, although those values may still appear in Table 1.

Full Sudoku-Extreme evaluation. The 1.02-TFLOP Sudoku entry in Table 4 is the separately measured Sudoku-Lite operating point. We additionally evaluate FRM on all $422{,}786$ unique puzzles in the official Sudoku-Extreme test split. The 275K checkpoint uses 4,096 NFE per trajectory and at most 1,024 sequential attempts. A symmetric-KL threshold fixed on held-out validation data accepts converged trajectories; if the attempt cap is reached, inference returns the evaluated prediction with the lowest residual. Replaying prefixes of the cached trajectories, breadth 38 solves $421{,}902$ puzzles (99.7909%) at 4.11 mean TFLOPs per puzzle, matching EqR's reported 99.79% while using $51\times$ fewer FLOPs than its breadth-128 configuration (210.0 TFLOPs). At breadth 1,024, FRM solves $422{,}392/422{,}786$ puzzles for 99.91% accuracy at 8.08 mean TFLOPs per puzzle. Target solutions are used only for final scoring.

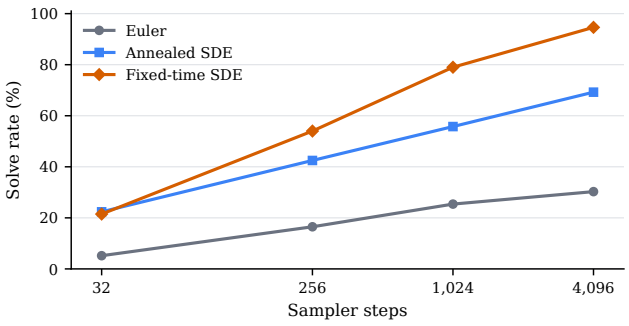
Table 4: **Best measured exact-solve accuracy and absolute inference compute.** Accuracy and mean TFLOPs per puzzle. Sudoku uses the 2,048-example Sudoku-Lite subset; Table 1 reports complete-test accuracy. † marks published values that were not reproduced in our evaluation; dashes mark unavailable compatible measurements.

	Sudoku-Lite		Zebra		Maze-Unique	
Method	**Acc. (%)**	**TFLOPs**	**Acc. (%)**	**TFLOPs**	**Acc. (%)**	**TFLOPs**
Diffusion language models and flows						
FMLM (Lee et al., 2026)	2.4	0.005	74.2	0.058	87.5	0.305
MDLM (Sahoo et al., 2024)	5.3	1.11	76.9†	–	89.0	9.22
ReMDM (Wang et al., 2025b)	7.7	2.21	–	–	93.5	9.22
FLM (Lee et al., 2026)	17.4	1.84	67.9	13.25	93.3	18.42
Adaptive MDLM (Kim et al., 2025)	22.0	2.21	98.3†	–	100.0	2.32
Specialized recurrent reasoners						
HRM (Wang et al., 2025a)	64.0	6.87	–	–	0.3	0.0454
TRM (Jolicoeur-Martineau, 2025)	93.9	36.0	–	–	77.9	0.113
FPRM (Movahedi et al., 2026)	88.6	47.69	51.3	12.62	100.0	1.80
SE-RRM (Freinschlag et al., 2026)	93.7	70.96	–	–	52.2	39.96
EqR (Huang et al., 2026)	99.2	210.0	–	–	93.0	111.81
Flow Reasoning Model (ours)	**99.9**	**1.02**	**100.0**	**0.071**	**100.0**	**0.170**

Adaptive computation. Token-adaptive methods can incur a nonzero minimum cost even at their lowest nominal setting. We therefore plot measured realized FLOPs rather than the maximum-step label.

B.2 Choice of Sampler

We evaluate one validation-selected, variable-time FPF checkpoint on the same 1,024 difficulty-balanced Sudoku-Extreme puzzles under three inference dynamics. Euler and annealed SDE integrate from noise to data over a descending time schedule; fixed-time SDE instead holds the physical corruption level at $t = 0.4$ and repeatedly applies the denoise–renoise map. All three use self-conditioning and the same four sampler-step budgets.



Sampler	Best (%)	Steps
Euler	30.3	4,096
Annealed SDE	69.2	4,096
Fixed-time SDE	**94.6**	**4,096**

Figure 7: **Fixed-time sampling unlocks recurrent scaling on Sudoku-Extreme.** The held-time sampler scales most strongly and reaches 94.6%, compared with 69.2% for annealed SDE and 30.3% for Euler. The table reports each sampler's best measured point; charged NFE is sampler steps plus one final readout.

We next sweep the held level itself while leaving the model, examples, sampler seed, and 4,096-step budget unchanged. Figure 8 shows a clear intermediate optimum.

	Sudoku-Lite		Zebra		Maze-Unique	
Inference policy	**Solve (%)**	**TFLOPs**	**Solve (%)**	**TFLOPs**	**Solve (%)**	**TFLOPs**
Fixed-depth FRM	98.34	3.670	100.00	4.180	99.50	2.913
+ Residual early stopping	98.10	0.359	100.00	0.071	99.60	0.105
+ Sequential restarts	99.90	1.019	100.00	0.071	100.00	0.170

Table 5: **Residual-guided inference operating points across three reasoning tasks.** Entries are exact solve rate and realized mean TFLOPs per problem for a matched inference-policy ablation. Sudoku uses the fixed 2,048-example Sudoku-Lite subset, Zebra the first 1,024 examples of its mixed-size test split, and Maze-Unique its complete 1,000-example test split. Each column uses one fixed checkpoint and inference seed, rather than averaging training seeds.

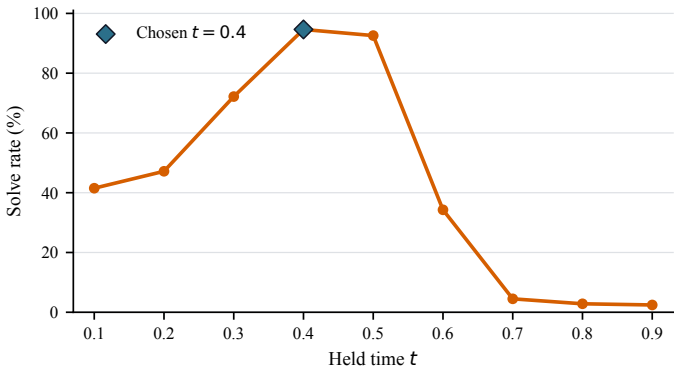


Figure 8: **An intermediate held time best supports recurrent refinement.** At 4,096 sampler steps, the selected $t = 0.4$ achieves the highest solve rate (94.6%), with $t = 0.5$ close behind (92.6%). Both lower and higher held levels underperform. Charged NFE is 4,097 after the final readout.

B.3 Residual-Guided Inference Across Tasks

We isolate the inference policy using one fixed FRM checkpoint and one fixed test set per task. The first row runs one held-time trajectory to its depth cap. The second stops that same first trajectory when a trailing-window Euclidean residual remains below a calibrated threshold and its decoded prediction is stable. The third sequentially restarts only trajectories that reach the depth cap without acceptance. If the breadth cap is exhausted, it returns the evaluated prediction with the lowest terminal residual. No target solution is consulted by the stopping, restart, or selection rule.

For Sudoku, we use a 4,096-NFE per-attempt cap, breadth 128, and a threshold selected on a separate 1,024-puzzle development set. Zebra and Maze-Unique use a 257-NFE cap and breadth 16; their thresholds are selected on fixed, test-disjoint samples of 256 puzzles from the corresponding official training split. Because these calibration puzzles come from the models' training splits, we do not characterize them as held-out validation data. Zebra accepts every puzzle on its first trajectory, so the stopping and restart rows coincide. On Maze-Unique, restarts recover the four errors remaining under early stopping and reach 100% exact solve at 0.170 mean TFLOPs.

To test whether the residual itself transfers across tasks, we pair each checked state with its exact-solve label and report the resulting state-level AUROC.

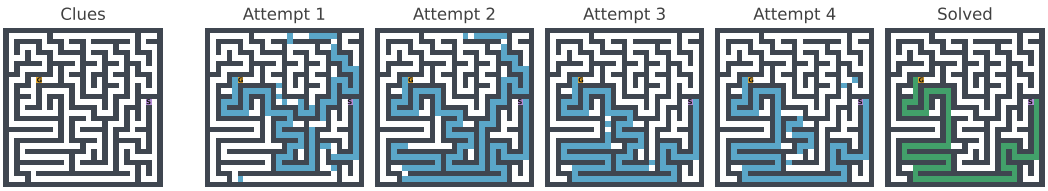


Figure 9: **Recurrent refinement solves a Maze-Unique instance.** From the fixed maze layout, start, and goal (left), the FRM progressively revises its proposed path (blue) before reaching the solution (green). Exact-grid error decreases monotonically from 140 to 87, 32, 17, and zero.

Dataset	FPF residual AUROC
Sudoku-Extreme	0.998
Zebra	1.000
Maze-Unique	0.989

Table 6: **FPF residual AUROC across reasoning tasks.** The Euclidean convergence residual strongly predicts exact correctness on all three datasets. The Sudoku value uses the fixed 2,048-puzzle Sudoku-Lite evaluation subset.

B.4 Training-Recipe Results

Table 2b reports three-seed means and standard deviations for every dataset. We report aggregate statistics consistently and omit individual-seed values.

C Additional Dynamics Analysis

We distinguish productive refinement from convergence to an incorrect fixed point using the following depth-dependent diagnostics.

Depth-dependent performance and distance to the solution. At each depth, we record exact solve rate and token-averaged cross-entropy to the one-hot solution on the same pinned examples. Thus changes isolate additional recurrent computation. In Fig. 6, FPF improves accuracy while driving ground-truth loss downward; conventional self-conditioning plateaus while its loss to the correct target grows.

Convergence residual and correctness prediction. For adjacent predictive states p_k and p_{k-1}, we use the token-averaged Euclidean residual defined above and pair its negative with the decoded state's exact-solve label. A near-zero residual indicates convergence, not necessarily correctness. The right panel of Fig. 6 uses 256 matched seed-0 Sudoku-Extreme examples at fixed $t = 0.4$. At the largest depths, FPF solves nearly every example, leaving few incorrect examples for estimating AUROC; we therefore interpret the persistence of the tail qualitatively.

Interpretation. Exact solve rate remains primary; loss-to-gold and adjacent-state residuals reveal whether depth corrects the solution or settles without reaching the correct target.

C.1 Two Axes of Test-Time Scaling

FRMs expose two complementary inference-time controls. Recurrent *depth* gives one trajectory more opportunities to refine its prediction, while stochastic *breadth* explores independent trajectories and uses the convergence residual to identify a stable candidate. Figure 10 varies both caps for the selected 275K-step 7M FRM on the same 2,048 Sudoku-Lite puzzles.

Depth removes most errors initially, but its returns diminish near saturation. At breadth one, increasing depth from 512 to 4,096 reduces the failure rate from 7.81% to 1.66%, while doubling depth again reaches 1.51%. Breadth continues to resolve these remaining failures: at depth 512, increasing

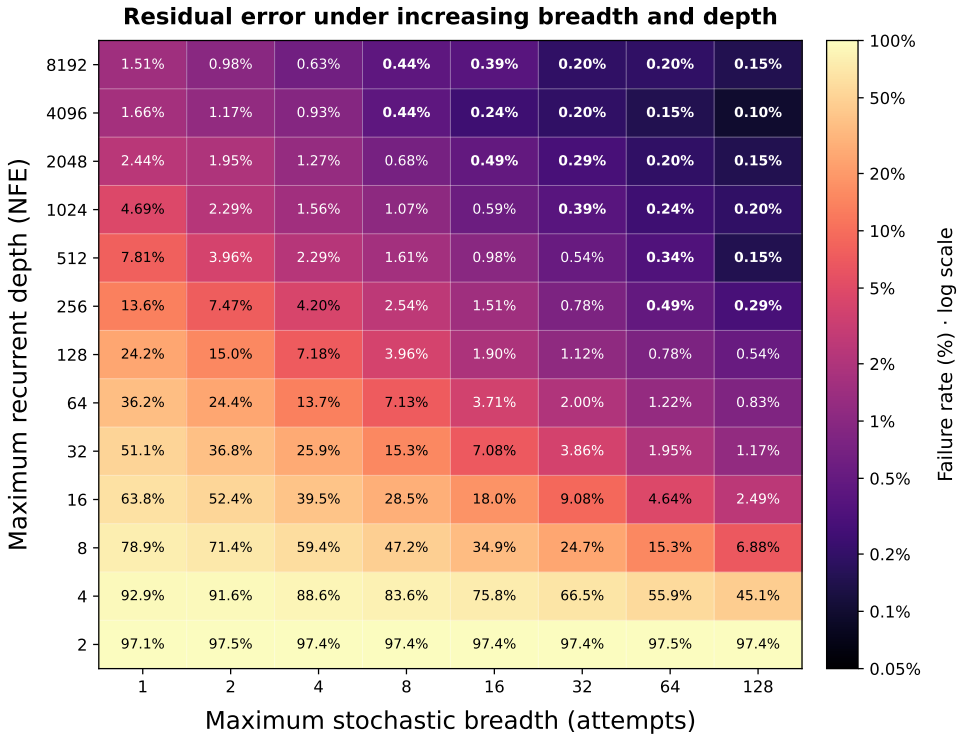


Figure 10: **Depth and breadth remove complementary failure modes.** Increasing recurrent depth sharply reduces failures within an attempt, while additional breadth (independent attempts, implemented as restarts after the first) continues to resolve the small residual error set near saturation. Failure rate is shown on a logarithmic color scale.

breadth from one to 16 lowers failure from 7.81% to 0.98%, and breadth 128 reaches 0.15%. Thus depth improves refinement within a trajectory, whereas breadth is especially useful for escaping the small set of failures that remain after deep recurrence. Because residual stopping determines the number of evaluations actually executed, this sweep compares policy caps rather than equal-FLOP allocations.

C.2 Residual Metric Robustness

We test whether the relationship between convergence and correctness depends on the particular residual metric. Using the same stochastic pass@1 trajectories from the 7M FRM at the 275k checkpoint on EqR's 2,048-puzzle Sudoku-Lite subset ($t = 0.4$, 4,096 NFE), we compute three label-free distances between adjacent predictions over the free cells and average each distance over the final fifth of recurrent transitions. The resulting evaluation contains 2,014 correct and 34 incorrect predictions. Ground-truth labels are used only after inference to determine exact correctness and calculate AUROC.

Adjacent-state metric	**AUROC**
Euclidean distance	0.9993 ± 0.0005
Symmetric KL	0.9993 ± 0.0005
Predictive cross-entropy	0.9849 ± 0.0145

Table 7: **Residual-metric robustness.** Lower residual predicts an exact solve. Values are the point estimate $\pm$ one bootstrap standard deviation from 10,000 stratified puzzle resamples.