SELF-CONDITIONING ENABLES DEEP SEQUENTIAL REASONING IN FLOW MATCHING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent work has explored visual reasoning through image generation, where models solve visual-reasoning problems by directly generating solution images. Intuitively, iterative samplers such as flow matching are well suited to this task, since each sampling step may resemble one reasoning step. However, in this work, from a circuit complexity perspective, we show that this intuition is misleading. We prove that as long as the model outputs satisfy certain mild regularity conditions, the trajectory of the vanilla flow-matching sampler can be simulated by a constant number of sequential rounds of parallel model evaluations. Consequently, additional sampling steps cannot increase the expressive power of a constant-depth backbone model under the standard complexity-theoretic model of transformers. We then reveal that *self-conditioning*, which conditions the model on the previous step's estimate of the final image, plays a previously unrecognized role in removing this bottleneck. Controlled experiments on diverse visual-reasoning tasks show that flow matching with self-conditioning achieves higher accuracy and greater reasoning depth than vanilla flow matching, and its performance improves with more sampling steps. Theoretically, we prove that with self-conditioning, a constant-depth backbone can achieve expressive power equivalent to that of space-bounded Turing machines. Our study suggests a design principle for future visual reasoning research: carry more information between sampling steps, so that additional steps translate into deeper reasoning.[1]

1 INTRODUCTION

Recent advances in visual generative models have sparked growing interest in *generative visual reasoning*, where models plan and reason directly in image space (Su et al., 2025b; Hu et al., 2024; Tong et al., 2026). This paradigm is natural for spatial planning, combinatorial optimization, and structured rendering (He et al., 2025; Dai et al., 2026). Intuitively, flow matching (Lipman et al., 2022; Liu et al., 2022) is well suited to this setting: it turns noise into an image through K evaluations of a velocity model, providing K sequential rounds for reasoning to unfold. He et al. (2025) put this intuition into practice, solving visual reasoning tasks via flow-matching image generation.

However, in this work we show that this apparent reasoning depth is an illusion. Through *circuit complexity* analysis, we find that the expressive power of vanilla flow matching (VANILLAFM) does not grow with the number of sampling steps K. The intuition lies in how flow matching generates images: the sampling process discretizes an ordinary differential equation (ODE), making only a small incremental update of scale $O(1/K)$ to its state at each step. Because the full trajectory is governed by this continuous ODE, we show that it can be simulated by an alternative ODE solver in a constant number of steps, regardless of how large K is. Consequently, running VANILLAFM with a constant-depth backbone leaves its expressive power collapsed to constant reasoning depth (§3.1).

To overcome this barrier, we uncover a previously unrecognized role of *self-conditioning* (Chen et al., 2022). In flow matching with self-conditioning (SELFCONDFM), the model still simulates the ODE, but its input is augmented with a prediction of its own sampling trajectory's endpoint, i.e., an estimate of the final image (§3.2). This endpoint prediction acts as an internal scratchpad on which the model records and refines intermediate partial solutions. Unlike the continuous state,

[1] We will release our code publicly upon publication.

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.

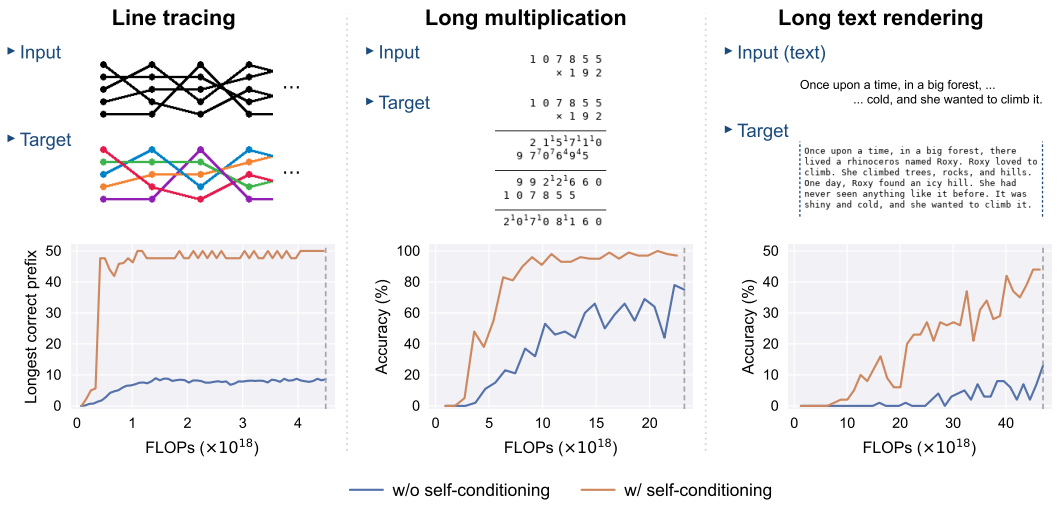


Figure 1: Controlled experiments on three synthetic tasks: line tracing, long multiplication, and long text rendering (§4.2). The top row shows an example input and target output for each task, and the bottom row plots performance against training FLOPs. Across all three tasks, self-conditioning improves the performance of flow matching by large margins.

which changes only by $O(1/K)$, this scratchpad can change abruptly between steps. Therefore, the resulting trajectory can no longer be simulated by an ODE solver in a constant number of steps. This prevents the collapse of expressive power, enabling deep sequential reasoning (§3.3).

We establish this advantage of self-conditioning both theoretically and empirically. Theoretically, we prove that self-conditioning can lift the expressive power of flow matching from constant-depth circuits to space-bounded Turing machines (§3). Empirically, controlled experiments on diverse synthetic tasks demonstrate that self-conditioning substantially improves the performance of flow matching (Figure 1), and its advantage also extends to broader visual-reasoning tasks (§4). Our contributions are as follows:

- **Self-conditioning as an internal scratchpad.** We reveal a previously unrecognized role of self-conditioning in generative reasoning: rather than serving merely as an empirical regularizer, its endpoint prediction provides an internal scratchpad that carries rich information, enabling deep sequential computation across sampling steps.
- **Circuit complexity separation.** We formalize this mechanism through circuit complexity analysis. We prove that as long as the model outputs satisfy certain mild regularity conditions, VANILLAFM with an AC^0 or TC^0 backbone collapses to constantly many sequential model evaluations, deciding only languages in the backbone's circuit complexity class regardless of the number of sampling steps (§3.1). In contrast, SELFCONDFM enables an AC^0 model to simulate space-bounded Turing machines, deciding P/poly with polynomial steps and $\mathsf{PSPACE}/\mathsf{poly}$ with at most exponential steps (§3.3). These results extend to more general settings (§3.4).
- **Empirical validation.** We support these theoretical results through controlled experiments on three synthetic tasks with known circuit complexity: NC^1-complete line tracing, and TC^0-complete long multiplication and long text rendering (§4.2). Under matched training FLOPs, SELFCONDFM outperforms VANILLAFM by large margins (Figure 1). Moreover, in agreement with our theory, SELFCONDFM keeps improving with more sampling steps, whereas VANILLAFM saturates early (Figure 3). Self-conditioning also brings consistent improvements on five broader visual-reasoning tasks from He et al. (2025) (§4.3).

2 PRELIMINARIES

We first review flow matching and circuit complexity (§2.1), then formalize the modeling conventions and regularity conditions underlying our theoretical analysis (§2.2).

2.1 BACKGROUND

Flow matching. Flow matching (Lipman et al., 2022; Liu et al., 2022) is a generative modeling framework. Given a data distribution p_{data} and a prior p_{prior}, we define the probability path p_t as the distribution of $\tilde{x}_t := tx_{\text{data}} + (1-t)\epsilon$ for independent random vectors $x_{\text{data}} \sim p_{\text{data}}$ and $\epsilon \sim p_{\text{prior}}$. The velocity field that generates this path is given by $v(x,t) := \mathbb{E}_{(x_{\text{data}},\epsilon)\sim p(\cdot|\tilde{x}_t=x)}[x_{\text{data}} - \epsilon]$. Flow matching aims to find a parameterized field $u_\theta(x,t)$ to approximate $v(x,t)$, so that the solution of the ODE $\frac{\mathrm{d}x}{\mathrm{d}t} = u_\theta(x,t)$ with $x|_{t=0} \sim p_{\text{prior}}$ follows the marginals p_t for $t \in [0,1]$.

Flow-matching models are trained by minimizing the conditional flow-matching loss $\mathcal{J}_{\text{CFM}} := \mathbb{E}_{t\sim\mathcal{U}[0,1],\epsilon\sim p_{\text{prior}},x_{\text{data}}\sim p_{\text{data}}}\|u_\theta(\tilde{x}_t,t) - (x_{\text{data}} - \epsilon)\|_2^2$. Lipman et al. (2022) prove that minimizing $\mathcal{J}_{\text{CFM}}$ is equivalent to minimizing $\mathbb{E}_{t,\epsilon,x_{\text{data}}}\|u_\theta(\tilde{x}_t,t) - v(\tilde{x}_t,t)\|_2^2$, so the trained model regresses onto the target field v. At inference time, a sample is generated by solving the ODE with a numerical scheme such as Euler's method (Hairer et al., 1993; Liu et al., 2022; Esser et al., 2024).

For conditional generation, such as text-to-image and image editing, the model is conditioned on context ctx, with p_{data} now a joint distribution over pairs $(\mathrm{ctx}, x_{\text{data}})$. This amounts to learning a velocity field $u_\theta(x,t;\mathrm{ctx})$ minimizing $\mathcal{J}_{\text{cond}} := \mathbb{E}_{t,\epsilon,(\mathrm{ctx},x_{\text{data}})}\|u_\theta(\tilde{x}_t,t;\mathrm{ctx}) - (x_{\text{data}} - \epsilon)\|_2^2$.

Circuit complexity. We measure the expressive power of a model by the complexity of the languages it can decide. A *language* is a subset $\mathcal{L} \subseteq \{0,1\}^*$, and to *decide* $\mathcal{L}$ is to determine whether a given input $w \in \{0,1\}^m$ belongs to $\mathcal{L}$. We abstract the model as a *circuit family* $(\mathcal{C}_m)_{m\geq 1}$, where each $\mathcal{C}_m$ is a Boolean circuit that handles inputs of length m (Arora et al., 2009). The *size* of a circuit is the number of gates and its *depth* is the length of the longest input-to-output path. Depth measures sequential computation, whereas size measures total work.

For $k \geq 0$, NC^k, AC^k, and TC^k denote the non-uniform classes of languages decided by polynomial-size, depth-$O(\log^k m)$ circuit families with bounded-fan-in Boolean, unbounded-fan-in Boolean, and unbounded-fan-in Boolean plus MAJORITY gates, respectively. We call circuits of unbounded-fan-in AND, OR, NOT, and MAJORITY gates *majority circuits*. The class $\mathsf{NC} := \bigcup_{k\geq 0}\mathsf{NC}^k$ comprises the efficiently parallelizable problems. The *advice classes* $\mathsf{P/poly}$ and $\mathsf{PSPACE/poly}$ consist of languages decided by polynomial-time and polynomial-space Turing machines, respectively, with polynomial-length advice depending only on the input length. Standard relations among these classes, such as $\mathsf{NC} \subseteq \mathsf{P/poly} \subseteq \mathsf{PSPACE/poly}$, are collected in Appendix A.1.

These classes bound the power of common neural network architectures: a constant-depth log-precision transformer without chain-of-thought (CoT) lies in TC^0 (Merrill et al., 2022; Merrill & Sabharwal, 2023; Chiang, 2024), and variants such as unique hard-attention transformers (Hao et al., 2022) and constant-precision transformers (Li et al., 2024) lie in AC^0 instead. Surpassing these bounds requires more layers or, as with CoT, additional sequential calls (Li et al., 2024). This motivates studying whether flow matching, which sequentially invokes such a network many times, extends what the model can decide.

2.2 THEORETICAL SETUP AND NOTATION

To formally analyze the expressive power of flow matching, we establish our theoretical setup below, adapting the flow-matching framework into a concrete computational model.

Modeling flow matching. We consider flow matching under finite-precision arithmetic with b fractional bits. The representable numbers form $F_b := \{2^{-b}k : k \in \mathbb{Z}\}$. All arithmetic operations are evaluated over the reals and then rounded coordinate-wise via $\mathrm{round}_b(x) := 2^{-b}\lfloor 2^b x + 1/2 \rfloor$.

A *model* u is a map acting on binary encodings of arbitrary length. It receives an encoded triple $(x,t,\mathrm{ctx}) \in F_b^n \times F_b \times F_b^*$, where n, b, and the length of ctx are non-negative integers read from the encoding, and outputs the velocity $u(x,t;\mathrm{ctx}) \in F_b^n$. On an input problem $w \in \{0,1\}^m$, vanilla flow matching $\textsc{VanillaFM}(w;u,K)$ sets $\mathrm{ctx} = w$ and runs the Euler method for K steps, starting from $\tilde{x}_0 \sim p_{\text{prior}}$, where p_{prior} is an arbitrary prior distribution whose support we additionally restrict to $[-1,1]^n$ (Algorithm 1).

Model regularity and acceptance. A model u is *B-bounded* if $\|u(x,t;\mathrm{ctx})\|_\infty \le B$ for all $x \in F_b^n$, $t \in [0,1] \cap F_b$, and $\mathrm{ctx} \in F_b^*$, and *L-Lipschitz continuous* if $\|u(x,t;\mathrm{ctx}) - u(x',t;\mathrm{ctx}')\|_\infty \le L\max\{\|x-x'\|_\infty, \|\mathrm{ctx}-\mathrm{ctx}'\|_\infty\}$ for all $x, x' \in F_b^n$, $t \in [0,1] \cap F_b$, and equal-length $\mathrm{ctx}, \mathrm{ctx}' \in F_b^*$. The Lipschitz condition is joint in (x, ctx) and uniform over t. For simplicity we measure Lipschitzness in the ℓ_∞ norm, but the analysis carries over to any ℓ_p norm; see §3.4.

Given an inference algorithm $\mathcal{A}$ and a constant $c > 0$ independent of m, we say that u *accepts a language $\mathcal{L}$ with margin c under $\mathcal{A}$* if, for every m and $w \in \{0,1\}^m$, the output $x_K = \mathcal{A}(w; u, K)$ satisfies, with probability 1 over the randomness in $\mathcal{A}$, $(x_K)_j \ge c$ for all coordinates j when $w \in \mathcal{L}$ and $(x_K)_j \le -c$ for all j when $w \notin \mathcal{L}$. Acceptance with a margin represents a robust decision, in which every output coordinate is separated from the threshold 0 by at least c. For asymptotic analysis, the precision b, state dimension n, and step count K may grow with the problem size m.

3 Self-Conditioning Increases Expressive Power

In this section, we first show that, under the mild regularity conditions below, vanilla flow-matching inference is confined to constant-depth circuit classes (§3.1), which restricts the problems it can solve. We then consider self-conditioning (§3.2), a simple mechanism that allows the same models to attain substantially greater expressive power (§3.3).

3.1 Vanilla Flow Matching Is Confined to Constant Depth

Although VanillaFM makes K sequential model evaluations, as long as the model outputs satisfy certain mild regularity conditions, **this sequential depth is an illusion**: the same decision can be reached in $O(L)$ sequential rounds of parallel evaluations of the model, regardless of how large K is. When L is a constant, these are constantly many rounds, so the decided language stays within the model's own complexity class. Below we view K as a function of the problem size m, and state the theorem under the setup of §2.2.

Theorem 1 (Upper bounds on the expressiveness of vanilla flow matching)**.** *Let $k \ge 0$ be a constant integer, $\mathsf{D} \in \{\mathsf{NC}^{k+1}, \mathsf{AC}^k, \mathsf{TC}^k\}$ be a circuit complexity class, u be a model that can be computed by circuits in D, and $\mathcal{L}$ be any language that can be accepted by u with margin 1 under* VanillaFM*. Assume that for some constants $B > 0$ and $L \ge 1$ independent of m, the model u is B-bounded and L-Lipschitz continuous. Then as long as the state dimension satisfies $n = \mathrm{poly}(m)$, and the number of fractional bits satisfies*

$$\big(1 + \Omega(1)\big)\log_2 K(m) \le b(m) \le O(\log m), \tag{1}$$

we have $\mathcal{L} \in \mathsf{D}$.

See Appendix A.2.4 for the proof. The case $k = 0$ recovers the canonical constant-depth classes: under vanilla flow matching, log-precision softmax-attention transformers in TC^0 (Merrill et al., 2022; Merrill & Sabharwal, 2023; Chiang, 2024) remain in TC^0, and the unique hard-attention and constant-precision variants in AC^0 (Hao et al., 2022; Li et al., 2024) remain in AC^0. Thus, ODE iteration alone adds no expressive power in this setting.

The core technique behind the proof is a parallel Picard iteration (Hairer et al., 1993). The ODE under which VanillaFM samples can amplify an initial error by a factor of e^L. A naive simulation that reruns the Euler method with fewer steps to reproduce the full K-step trajectory therefore takes $\mathrm{e}^{O(L)}$ steps to reach constant accuracy. We instead iteratively update all K positions in parallel using the previous iterate, making the propagated error decay at a factorial rate across rounds up to an additive floor, so that $O(L)$ rounds suffice. This yields the following lemma.

Lemma 1. *Under the same setup as Theorem 1, assume that u is B-bounded for a constant $B > 0$ and $L(m)$-Lipschitz continuous with $L(m) \ge 1$. Then there exist a constant $c_0 = c_0(B) > 0$, independent of m, and an inference algorithm that runs for $O(L(m))$ iterations, making at most $K(m)$ parallel calls to u per iteration, such that as long as the number of fractional bits satisfies*

$$b(m) \ge \log_2 K(m) + (\log_2 \mathrm{e})L(m) + c_0 \qquad \textit{for every } m, \tag{2}$$

u accepts $\mathcal{L}$ with margin $1/2$ under that algorithm.

Algorithm 1 Vanilla Flow Matching

function VANILLAFM($w; u, K$)
  $t_i \leftarrow \mathrm{round}_b(i/K)$ for $i = 0, \ldots, K$
  Sample $\tilde{x}_0 \sim p_{\text{prior}}$; $x_0 \leftarrow \mathrm{round}_b(\tilde{x}_0)$

  for $i = 0, \ldots, K-1$ **do**
    $v_i \leftarrow u(x_i, t_i; w)$
    $x_{i+1} \leftarrow x_i + (t_{i+1} - t_i) \cdot v_i$

  end for
  return x_K
end function

Algorithm 2 Self-Conditioning Flow Matching

function SELFCONDFM($w; u, K$)
  $t_i \leftarrow \mathrm{round}_b(i/K)$ for $i = 0, \ldots, K$
  Sample $\tilde{x}_0 \sim p_{\text{prior}}$; $x_0 \leftarrow \mathrm{round}_b(\tilde{x}_0)$
  $z_0 \leftarrow \mathbf{0}^n$
  for $i = 0, \ldots, K-1$ **do**
    $v_i \leftarrow u(x_i, t_i; \mathrm{concat}(w, z_i))$
    $x_{i+1} \leftarrow x_i + (t_{i+1} - t_i) \cdot v_i$
    $z_{i+1} \leftarrow x_i + (1 - t_i) \cdot v_i$
  end for
  return x_K
end function

See Appendix A.2.2 for the proof. For constant L, these are constantly many rounds, and the arithmetic accompanying the model calls is shallow enough to be absorbed into the model's own circuit class, yielding Theorem 1. The $O(L)$ bound also lets us extend the analysis to a growing L (§3.4).

3.2 SELF-CONDITIONING

The expressivity bottleneck in Theorem 1 stems from how information flows along the continuous ODE: consecutive steps in VANILLAFM transfer information solely through the state x_i, which changes only by a small increment scaled by a step size of $O(1/K)$: $x_{i+1} = x_i + (t_{i+1} - t_i)v_i$. This small ODE update is what allows the Picard iteration in Lemma 1 to compress all K sampling steps into $O(L)$ iterations. One potential way to overcome this bottleneck is to make a larger update per step, for example, by introducing an extra input component that can change abruptly between sampling steps.

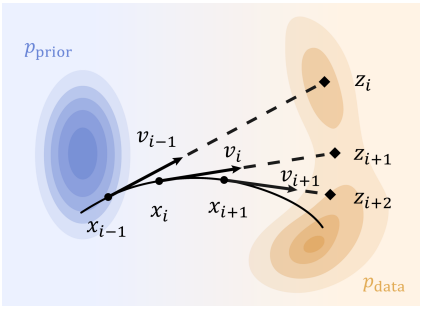


Figure 2: Overview of self-conditioning. Each prediction z_{i+1} is obtained by extrapolating the state x_i along its velocity v_i to $t = 1$.

We find that a simple mechanism achieves this: ***self-conditioning*** (Chen et al., 2022), originally proposed for discrete diffusion models and later adapted to other settings (Strudel et al., 2022; Stärk et al., 2023; Gupta et al., 2024). As stated in Algorithm 2, self-conditioning feeds the model an additional endpoint prediction $z_i \in F_b^n$, which is a linear approximation of x_{data}, obtained by extrapolating along the previous velocity: $z_i := x_{i-1} + (1 - t_{i-1})v_{i-1}$ for $i \geq 1$, with $z_0 = \mathbf{0}^n$.

For training, we use a two-pass heuristic inspired by prior work (Chen et al., 2022). Given $(w, x_{\text{data}}) \sim p_{\text{data}}$, $t \sim \mathcal{U}[0, 1]$, $\epsilon \sim p_{\text{prior}}$, and $\tilde{x}_t = t x_{\text{data}} + (1 - t)\epsilon$, let

$$v^{(1)} := u_\theta\big(\tilde{x}_t, t; \mathrm{concat}(w, \mathbf{0}^n)\big), \qquad v^{(2)} := u_\theta\big(\tilde{x}_t, t; \mathrm{concat}(w, \mathrm{sg}[\tilde{x}_t + (1 - t)v^{(1)}])\big),$$

where $\mathrm{sg}[\cdot]$ stops the gradient. The first pass imitates the initial sampling step, at which $z_0 = \mathbf{0}^n$, and the second imitates later steps. Both passes are weighted equally via the following objective:

$$\mathcal{J}_{\text{SC}} := \tfrac{1}{2}\,\mathbb{E}_{t,\epsilon,(w,x_{\text{data}})}\big[\|v^{(1)} - (x_{\text{data}} - \epsilon)\|_2^2 + \|v^{(2)} - (x_{\text{data}} - \epsilon)\|_2^2\big].$$

Consequently, by training on predictions that closely recover z_i, the model learns to leverage the actual z_i generated in the preceding sampling step, requiring only one extra forward pass during training. Figure 2 provides an overview of the mechanism.

Self-conditioning in practice. Self-conditioning is particularly natural for visual-space reasoning: the fed-back prediction z_i approximates the clean sample x_{data} and is therefore near the target data manifold in image space. Modern image generation models already accept images as conditions (Black Forest Labs et al., 2025; Wu et al., 2025), so it is reasonable to expect knowledge

from image-conditioned pretraining to transfer to self-conditioning, even though that pretraining involves no self-conditioning. The initialization $z_0 = \mathbf{0}^n$ is a placeholder for cleaner theoretical analysis. In actual implementation, for better efficiency, it is omitted both in the first training pass $v^{(1)} = u_\theta(\tilde{x}_t, t; w)$ and at the first sampling step, without affecting empirical performance.

3.3 SELF-CONDITIONING REACHES PSPACE/poly

We now formally show that self-conditioning expands the expressiveness of flow matching. Since the extra input z_{i+1} is a fresh prediction rather than an $O(1/K)$ increment on x_i, it is unconstrained by the step size and can change abruptly between sampling steps. Thus, the convergence behind Lemma 1 is lost, and the constant-depth barrier disappears altogether: **constant-depth models with self-conditioning can decide** P/poly **in polynomially many steps and** $\mathsf{PSPACE}/\mathsf{poly}$ **in at most exponentially many steps**.

Theorem 2 (Lower bounds on the expressiveness of flow matching with self-conditioning)**.** *For any language $\mathcal{L}$ in $\mathsf{PSPACE}/\mathsf{poly}$, there exists a model u that accepts $\mathcal{L}$ with margin 1 under* SELFCONDFM *and satisfies the following conditions: (a) u can be computed by AC^0 circuits; (b) u is 4-bounded and 8-Lipschitz continuous; (c) $b = O(\log m)$, $n = \mathrm{poly}(m)$, $K \le 2^{\mathrm{poly}(m)}$. If moreover $\mathcal{L} \in \mathsf{P}/\mathsf{poly}$, then the same result holds with* (1) *satisfied.*

The proof makes the sampler simulate a Turing machine with advice. The prediction z_i holds the machine's tape and state, and each model call advances the machine by one step, so K calls run the machine for $\Omega(K)$ steps. See Appendix A.3.2 for the construction.

In the P/poly case, all the regularity hypotheses of Theorem 1 are retained, so the two theorems concern the same model class and the gap comes from the inference algorithm alone. For $\mathsf{D} \in \{\mathsf{AC}^0, \mathsf{TC}^0\}$, the two samplers therefore **separate** across the barrier $\mathsf{P}/\mathsf{poly} \setminus \mathsf{D}$: given any language $\mathcal{L} \in \mathsf{P}/\mathsf{poly} \setminus \mathsf{D}$, Theorem 2 supplies an AC^0 model accepting it under SELFCONDFM, whereas Theorem 1 leaves no D model satisfying its hypotheses capable of accepting it under VANILLAFM. For $\mathsf{D} = \mathsf{AC}^0$, the separation always holds since $\mathsf{AC}^0 \subsetneq \mathsf{TC}^0$ (Furst et al., 1984); for $\mathsf{D} = \mathsf{TC}^0$, it holds under the conjecture $\mathsf{TC}^0 \subsetneq \mathsf{NC}^1$ (Vollmer, 1999), which implies that $\mathsf{TC}^0 \subsetneq \mathsf{P}/\mathsf{poly}$.

3.4 EXTENSIONS

Beyond the main results above, the analysis extends to more general settings. Lemma 1 directly implies an NC upper bound for VANILLAFM when the Lipschitz constant L grows as $L(m) = \mathrm{poly}(\log m)$, or TC^k upper bounds when further $L = O(\log^k m)$ for any constant integer $k \ge 0$ (Appendix B.1). Restricting u to be Lipschitz continuous under ℓ_p norms rather than ℓ_∞ norms preserves both Theorem 1 and Theorem 2 (Appendix B.2). Replacing the Euler method with any fixed explicit Runge–Kutta method (Hairer et al., 1993) whose nodes lie in $[0, 1]$, such as Heun's method (Karras et al., 2022), leaves the asymptotic results for VANILLAFM unchanged (Appendix B.3).

On the other hand, the $O(L(m))$ bound of Lemma 1 is tight with respect to circuit depth: any polynomial-size majority circuit of depth d can be simulated under VANILLAFM in $O(d)$ sequential rounds by an $O(d)$-Lipschitz continuous model u computed by TC^0 circuits. The bound of Lemma 1 therefore cannot be improved to $o(L(m))$ rounds of the same form unless the depth hierarchy for polynomial-size majority circuits collapses (Appendix B.4).

4 EXPERIMENTS

The analysis in §3 establishes an expressivity separation under the stated assumptions: SELFCONDFM can decide languages beyond the reach of VANILLAFM. Whether models trained on finite data realize this expressive advantage remains an empirical question. Therefore, we conduct controlled experiments to empirically validate the expressivity separation (§4.2). We further extend the evaluation to the five structured visual-reasoning tasks used by He et al. (2025) through controlled comparisons and larger-scale experiments (§4.3). Finally, we provide supporting analyses that assess the applicability of our theory to trained models and characterize the inference behavior of VANILLAFM and SELFCONDFM (§4.4).

4.1 Experimental Setup

Within each task, VanillaFM and SelfCondFM share the same backbone architecture, training dataset, optimizer, and noise schedule. The only difference is that SelfCondFM passes the endpoint prediction from one backbone model evaluation to the next as an additional condition (§3.2). During training, SelfCondFM makes two forward passes per example instead of one, costing extra compute, so we match the two methods on cumulative training FLOPs. Appendix D.1 provides the complete configurations and FLOPs accounting.

Since both VanillaFM and SelfCondFM make one backbone call per sampling step, we compare them by default at the same number of function evaluations (NFE), which directly measures sequential computation and equals the step count K in Algorithms 1 and 2. Unless otherwise stated, we use 50 NFEs; the NFE-scaling study in §4.2 varies the number of evaluations.

4.2 Empirical Validation of the Expressivity Separation

To empirically validate the expressivity separation, we first compare the two methods on three tasks with natural step-by-step solutions: *line tracing*, *long multiplication*, and *long text rendering* (Figure 1). Each task, formally defined and analyzed in Appendix C, has circuit complexity matching the separation barrier identified in §3.1 for either AC^0 or TC^0 backbones (the latter case assumes $\mathsf{TC}^0 \neq \mathsf{NC}^1$). Consequently, we expect these tasks to lie beyond the expressive power of vanilla flow matching with the corresponding backbone.

Tasks. We denote the problem scale for each task by m. Line tracing (NC^1-complete) requires coloring five long polylines that go through m permutation stages; long multiplication (TC^0-complete) entails completing a step-by-step long multiplication worksheet to compute the product of two integers, each with at most m digits; and long text rendering (TC^0-complete) requires rendering some paragraphs containing a total of m words onto a fixed-width canvas. We compare the methods on finite-scale synthetic datasets. For line tracing, we report the mean longest correct prefix (LCP), which is the average number of initial stages through which all five polylines remain correctly colored; for the other tasks, we report instance-level accuracy. See Appendix D.2 for more details.

Backbone architectures. All three tasks use `FLUX.2`-based architectures with their pretrained VAE kept frozen for image encoding and decoding (Black Forest Labs, 2026). For line tracing, we train a model from scratch; for the other two tasks, we fine-tune `FLUX.2-klein-base-4B` to leverage its prior knowledge of character recognition and generation.

Results. Figure 1 plots performance against training FLOPs. On line tracing, both methods are evaluated at 100 NFEs. SelfCondFM reaches the maximum LCP of 50 at 1.10×10^{18} training FLOPs, while VanillaFM plateaus near 8. On long multiplication, SelfCondFM exceeds 90% accuracy within half the budget, a level VanillaFM never reaches over the evaluated compute range. On long text rendering, SelfCondFM's accuracy rises with training compute, while VanillaFM's stays near zero for most of training. Self-conditioning thus **converges faster and performs better on all three tasks**, consistent with our theoretical separation. See Appendix D.2.1 for multi-seed evaluation and Appendix D.2.2 for generated examples.

NFE scaling. We further examine how the two methods use additional inference steps. Our theory in §3 suggests a contrasting behavior: under its assumptions, VanillaFM is confined to constant reasoning depth, whereas SelfCondFM can realize sequential reasoning depth that grows with the number of sampling steps. We test whether this predicted difference emerges on the line tracing task by varying the NFE.

We select two checkpoints with matched training FLOPs, and evaluate both at different NFEs. Figure 3 shows that both methods begin with a mean LCP of 0 at NFE $= 1$, but diverge as NFE increases. SelfCondFM reaches

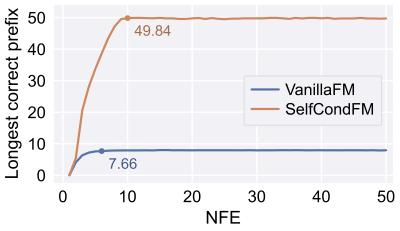


Figure 3: SelfCondFM's performance scales strongly with NFE, while that of VanillaFM saturates early.

Table 1: Controlled experiments on the DiffThinker tasks (accuracy, %). Under matched training FLOPs and NFE, SELFCONDFM outperforms VANILLAFM on every task.

Method	VSP-Super	Maze	TSP	Sudoku	Jigsaw	Avg.
VANILLAFM	89.0	64.3	51.0	60.5	64.7	65.9
SELFCONDFM	**92.0** (+3.0)	**70.7** (+6.4)	**66.7** (+15.7)	**80.0** (+19.5)	**74.3** (+9.6)	**76.7** (+10.8)

Table 2: Large-scale comparison on the DiffThinker tasks (accuracy, %). Each column header is a difficulty level, following the evaluation of Dai et al. (2026), from which the baseline numbers are taken. The best results are in bold and the second best are underlined. For unified training, each method uses a single adapter trained jointly on data from all four tasks.

Method	VSP-Super		Maze			TSP			Sudoku				Avg.
	16	32	8	16	32	12	15	18	45	40	35	30	
Task-specific training													
Qwen3-VL-8B (SFT)	61	8	53	37	0	59	60	43	30	17	2	0	30.8
Qwen3-VL-8B (GRPO)	0	0	0	0	0	0	0	0	0	0	0	0	0.0
DiffThinker	99	80	**100**	**100**	65	76	72	59	97	94	55	13	75.8
EndoCoT	99	85	**100**	**100**	90	77	**77**	**73**	**100**	**100**	95	64	88.3
Ours	**100**	**86**	**100**	**100**	**95**	**79**	74	72	**100**	**100**	**98**	**76**	**90.0**
Unified training													
DiffThinker	97	84	98	99	66	64	49	34	94	45	26	37	66.1
EndoCoT	**100**	80	97	98	52	64	55	46	**100**	88	80	59	76.6
Ours	**100**	**86**	**100**	**100**	**96**	**78**	**75**	**65**	**100**	**100**	**97**	**85**	**90.2**

49.84 at NFE $= 10$, whereas VANILLAFM reaches 7.66 at NFE $= 6$ and remains below 8.0 through NFE $= 50$. The extra steps thus yield deeper reasoning for SELFCONDFM, confirming that self-conditioning lets the model make better use of more sampling steps. Full details and inference-FLOP-aligned comparisons are provided in Appendix D.3.

4.3 EVALUATION ON STRUCTURED VISUAL-REASONING TASKS

We next investigate whether the gains observed in §4.2 extend to the five structured visual-reasoning tasks of He et al. (2025): large-scale visual spatial planning (VSP-Super), Maze, the traveling salesperson problem (TSP), Sudoku, and Jigsaw. See Appendix D.4 for detailed setup and illustrations.

Controlled experiments. Following the setup in §4.1, we train VANILLAFM and SELFCONDFM as `FLUX.2-klein-base-4B` LoRA adapters on the same data for each task, and evaluate them at matched training FLOPs and NFE.

As reported in Table 1, SELFCONDFM improves over VANILLAFM on every task, raising the average accuracy from 65.9% to 76.7%. Larger gains appear on TSP (+15.7%) and Sudoku (+19.5%), while notable improvements are also observed on the other tasks. Thus self-conditioning extends its advantage from our validation tasks to the task suite of He et al. (2025). Appendix D.4.2 shows some generated examples.

Large-scale comparisons. To evaluate the method at a larger scale, we fine-tune SELFCONDFM as a LoRA adapter on the 20B-parameter `Qwen-Image-Edit-2511` backbone (Qwen Team, 2025) following the setup of Dai et al. (2026), and compare it with the published baselines. Jigsaw is reported separately in Appendix D.4.1, because EndoCoT (Dai et al., 2026) does not evaluate it.

As shown in Table 2, task-specific SELFCONDFM adapters achieve 90.0% average accuracy, outperforming the purely visual-generative DiffThinker baseline (75.8%). EndoCoT (88.3%) additionally guides image generation using latent CoT in a multimodal language model, so we include it as a reference rather than a directly comparable baseline. A single adapter trained jointly on the four tasks reaches 90.2%, matching the per-task adapters and exceeding the unified DiffThinker (66.1%) and EndoCoT (76.6%) results by large margins.

4.4 Additional Empirical Analyses

Finally, we provide some empirical analyses to illustrate how the trained models behave during inference (detailed in Appendix D.5):

- **Lipschitzness.** We estimate the Lipschitz constant of a VanillaFM model trained on mixed-length line tracing instances. The estimate levels off at a roughly constant value of 400 for $m \geq 30$, consistent with the bounded-Lipschitz regime assumed in §3.1 (Appendix D.5.1).
- **Attention allocation.** The endpoint prediction z_i in SelfCondFM receives a nontrivial share of attention throughout the generation, with a larger share early on when x_i is still noisy. This indicates that the model indeed uses z_i to guide its generation (Appendix D.5.2).
- **Parallel and revisable reasoning.** Reasoning with flow matching has some intrinsic advantages, and SelfCondFM preserves them well. The model can form multiple local results in parallel within one trajectory and correct a wrong mid-trajectory generation in place (Appendix D.5.3).

5 Related Work

Generative visual reasoning. Multimodal large language models (MLLMs) typically treat images as static inputs and externalize reasoning through textual CoT. The emerging *thinking with images* paradigm instead turns vision into an intermediate workspace that can be repeatedly inspected or modified (Su et al., 2025b). Early approaches manipulate images through tool calls (Su et al., 2025a; 2026; Lai et al., 2026), while other approaches internalize visual intermediates within the model (Xu et al., 2026; Yang et al., 2026; Zhang et al., 2025) (see Appendix E.1 for more).

Recent work moves reasoning into image generation itself: DiffThinker uses direct image-to-image generation (He et al., 2025), while EndoCoT bridges latent states in an MLLM's CoT to image generation (Dai et al., 2026). However, the computational structure of generative visual reasoning remains underexplored. Liu (2025) establishes a TC^0 upper bound for optimal denoising diffusion probabilistic models, but requires the model to match the score function exactly, an assumption that rarely holds for practical trained models.

Self-conditioning. Self-conditioning feeds a model's endpoint prediction back into a subsequent denoising step. It was introduced by Chen et al. (2022) for diffusion over discrete data, and has since been adapted to continuous diffusion language modeling, image and video generation, and molecular flow matching (Strudel et al., 2022; Jabri et al., 2022; Gupta et al., 2024; Stärk et al., 2023). Prior work primarily studies generation quality and efficiency empirically; we instead treat self-conditioning as maintaining a computational state and analyze its expressive power.

6 Conclusion and Discussion

This work reveals a previously unrecognized role of self-conditioning in generative reasoning. We show that vanilla flow matching cannot deepen sequential reasoning through additional sampling steps. Self-conditioning breaks this bottleneck by passing endpoint predictions across steps, which lets additional sampling steps translate into reasoning depth. Experiments across diverse visual-reasoning tasks confirm this separation: self-conditioning achieves better performance and consistently benefits from more sampling steps.

These theoretical guarantees, however, rely on certain regularity assumptions. In particular, the upper bound in Theorem 1 requires a fairly strong Lipschitz restriction. Moreover, the Lipschitz estimates in §4.4 are relatively large and provide empirical rather than certified bounds (Appendix D.5.1). Since the dependence on the Lipschitz constant is tight up to constant factors (§3.4), future work may explore alternative forms of regularity conditions.

Although we focus on image-space visual reasoning, our theoretical analysis does not depend on the data modality. It therefore applies wherever flow matching serves as the sampling backbone, including video generation (Kong et al., 2024; Ifriqi et al., 2026), continuous diffusion language modeling (Chen et al., 2026), and visuomotor policy learning (Black et al., 2024). Whether self-conditioning delivers similar empirical gains remains an open question. Nevertheless, this perspective may inform future sampler design in settings where iterative generation is expected to reason.

AI DISCLOSURE

For this work, we used generative AI tools to (i) generate synthetic datasets for the tasks in §4.2; (ii) provide insights into reducing the number of sequential rounds from $\mathrm{e}^{O(L)}$ to $O(L)$ by replacing the Euler method with Picard iterations in §3.1; (iii) assist in writing the proofs; and (iv) implement the training codebase. We did not use generative AI tools to develop theoretical models or conceptual frameworks, formulate mathematical claims, design or provide feedback on research methodology or experiments, support qualitative or thematic data analysis, or interpret results. The tasks of proposing or refining hypotheses, translating text, and cleaning or reformatting datasets were not applicable to this work. Additionally, we used generative AI tools to create and refine figures, to assist in drafting and editing the paper at the paragraph level to improve readability, to identify relevant literature, and to format references.

We have reviewed all AI-assisted work. All AI-assisted proof writing was reviewed by at least two authors other than the original writers. All AI-assisted language editing was reviewed by at least one author other than those who originally drafted the corresponding text. The core experimental codebase and synthetic datasets were reviewed by at least one author. All figures, relevant literature, and the final reference formatting were verified by at least one author. We take full responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

For the theoretical results, §2.2 specifies the theoretical setup, and §3 states the assumptions of each theorem. Appendices A and B give complete proofs of the main theorems and their extensions, and Appendix C formally defines the three synthetic tasks in §4.2 and proves their circuit-complexity classifications. For the experiments, Appendix D.1 describes the backbones, the implementation of self-conditioning, and the training-FLOP accounting. Appendix D.2 details the data generation, training hyperparameters, and evaluation protocol for each synthetic task, Appendix D.2.1 reports results over multiple seeds, and Appendix D.3 elaborates the selection of checkpoints and the evaluation for our NFE-scaling study. Appendix D.4 describes the data preparation, evaluators, and training configurations for the structured visual-reasoning tasks in §4.3, which follow Dai et al. (2026) and He et al. (2025).

REFERENCES

El Mehdi Achour, François Malgouyres, and Franck Mamalet. Existence, stability and scalability of orthogonal convolutional neural networks. *Journal of Machine Learning Research*, 23(347): 1–56, 2022.

Miklós Ajtai. Approximate counting with uniform constant-depth circuits. *Advances in computational complexity theory*, 13:1–20, 1990.

Alireza Amiri, Xinting Huang, Mark Rofin, and Michael Hahn. Lower bounds for chain-of-thought reasoning in hard-attention transformers. *arXiv preprint arXiv:2502.02393*, 2025.

Sanjeev Arora, Boaz Barak, et al. *Computational complexity: a modern approach*, volume 1. Cambridge University Press Cambridge, 2009.

David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pp. 1–5, 1986. doi: 10.1145/12130.12131.

David A. Mix Barrington, Kevin J. Compton, Howard Straubing, and Denis Thérien. Regular languages in NC^1. *Journal of Computer and System Sciences*, 44(3):478–499, 1992. doi: 10.1016/0022-0000(92)90014-A.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Black Forest Labs. Flux.2 [klein] 4b base. https://huggingface.co/black-forest-labs/FLUX.2-klein-base-4B, 2026. Hugging Face model card.

Black Forest Labs, Stephen Batifol, Andreas Blattmann, Frederic Boesel, Saksham Consul, Cyril Diagne, Tim Dockhorn, Jack English, Zion English, Patrick Esser, et al. Flux. 1 kontext: Flow matching for in-context image generation and editing in latent space. *arXiv preprint arXiv:2506.15742*, 2025.

Ashok K Chandra, Larry Stockmeyer, and Uzi Vishkin. Constant depth reducibility. *SIAM Journal on Computing*, 13(2):423–439, 1984.

Bo Chen, Xiaoyu Li, Yingyu Liang, Jiangxuan Long, Zhenmei Shi, Zhao Song, and Jiahao Zhang. Circuit complexity bounds for rope-based transformer architecture. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 11091–11108, 2025.

Ting Chen, Ruixiang Zhang, and Geoffrey Hinton. Analog bits: Generating discrete data using diffusion models with self-conditioning. *arXiv preprint arXiv:2208.04202*, 2022.

Yuxin Chen, Chumeng Liang, Hangke Sui, Ruihan Guo, Chaoran Cheng, Jiaxuan You, and Ge Liu. Langflow: Continuous diffusion rivals discrete in language modeling. *arXiv preprint arXiv:2604.11748*, 2026.

Dongjie Cheng, Yongqi Li, Zhixin Ma, Hongru Cai, Yupeng Hu, Wenjie Wang, Liqiang Nie, and Wenjie Li. Omni-R1: Towards the unified generative paradigm for multimodal reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 38907–38924. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1937.

David Chiang. Transformers in uniform TC^0. *arXiv preprint arXiv:2409.13629*, 2024.

David Chiang, Peter Cholak, and Anand Pillay. Tighter bounds on the expressivity of transformer encoders. In *International Conference on Machine Learning*, pp. 5544–5562. PMLR, 2023.

Xuanlang Dai, Yujie Zhou, Long Xing, Jiazi Bu, Xilin Wei, Yuhong Liu, Beichen Zhang, Kai Chen, and Yuhang Zang. Endocot: Scaling endogenous chain-of-thought reasoning in diffusion models. *arXiv preprint arXiv:2603.12252*, 2026.

Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first international conference on machine learning*, 2024.

Merrick Furst, James B Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical systems theory*, 17(1):13–27, 1984.

Jiawei Gu, Yunzhuo Hao, Huichen Wang, Linjie Li, Michael Qizhe Shieh, Yejin Choi, Ranjay Krishna, and Yu Cheng. Thinkmorph: Emergent properties in multimodal interleaved chain-of-thought reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 141405–141447, 2026.

Agrim Gupta, Lijun Yu, Kihyuk Sohn, Xiuye Gu, Meera Hahn, Fei-Fei Li, Irfan Essa, Lu Jiang, and José Lezama. Photorealistic video generation with diffusion models. In *European Conference on Computer Vision*, pp. 393–411. Springer, 2024.

Ernst Hairer, Gerhard Wanner, and Syvert P Nørsett. *Solving ordinary differential equations I: Nonstiff problems*. Springer, 1993.

András Hajnal, Wolfgang Maass, Pavel Pudlák, Mario Szegedy, and György Turán. Threshold circuits of bounded depth. *Journal of Computer and System Sciences*, 46(2):129–154, 1993.

Yiding Hao, Dana Angluin, and Robert Frank. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Transactions of the Association for Computational Linguistics*, 10:800–810, 2022.

John Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pp. 6–20, 1986.

Zefeng He, Xiaoye Qu, Yafu Li, Tong Zhu, Siyuan Huang, and Yu Cheng. Diffthinker: Towards generative multimodal reasoning with diffusion models. *arXiv preprint arXiv:2512.24165*, 2025.

William Hesse, Eric Allender, and David A Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65(4): 695–716, 2002.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. Video diffusion models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 8633–8646, 2022.

Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, and Ranjay Krishna. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models. *Advances in Neural Information Processing Systems*, 37:139348–139379, 2024.

Tariq Berrada Ifriqi, John Nguyen, Karteek Alahari, Jakob Verbeek, and Ricky TQ Chen. Flowception: Temporally expansive flow matching for video generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16185–16195, 2026.

Allan Jabri, David Fleet, and Ting Chen. Scalable adaptive computation for iterative generation. *arXiv preprint arXiv:2212.11972*, 2022.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in neural information processing systems*, 35:26565–26577, 2022.

Akitoshi Kawamura. Lipschitz continuous ordinary differential equations are polynomial-space complete. *Computational Complexity*, 19(2):305–332, 2010.

Weijie Kong, Qi Tian, Zijian Zhang, Rox Min, Zuozhuo Dai, Jin Zhou, Jiangfeng Xiong, Xin Li, Bo Wu, Jianwei Zhang, et al. Hunyuanvideo: A systematic framework for large video generative models. *arXiv preprint arXiv:2412.03603*, 2024.

Xin Lai, Junyi Li, Wei Li, Tao Liu, Tianjian Li, and Hengshuang Zhao. Mini-o3: Scaling up reasoning patterns and interaction turns for visual search. In *International Conference on Learning Representations*, volume 2026, pp. 76722–76746, 2026.

Sangyun Lee, Zinan Lin, and Giulia Fanti. Improving the training of rectified flows. *Advances in neural information processing systems*, 37:63082–63109, 2024.

Bangzheng Li, Ximeng Sun, Jiang Liu, Ze Wang, Jialian Wu, Xiaodong Yu, Emad Barsoum, Muhao Chen, and Zicheng Liu. Latent visual reasoning. In *International Conference on Learning Representations*, 2026.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. In *International Conference on Learning Representations*, volume 2024, pp. 11911–11943, 2024.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

Yuxi Liu. Perfect diffusion is tc0–bad diffusion is turing-complete. *arXiv preprint arXiv:2507.12469*, 2025.

Edward James McShane. Extension of range of functions. *Bulletin of the American Mathematical Society*, 40(12):837–842, 1934.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023.

William Merrill, Ashish Sabharwal, and Noah A Smith. Saturated transformers are constant-depth threshold circuits. *Transactions of the Association for Computational Linguistics*, 10:843–856, 2022.

Kaleb Newman, Tyler Zhu, and Olga Russakovsky. Video models reason early: Exploiting plan commitment for maze solving. *arXiv preprint arXiv:2603.30043*, 2026.

Yiming Qin, Bomin Wei, Jiaxin Ge, Konstantinos Kallidromitis, Stephanie Fu, Trevor Darrell, and XuDong Wang. Chain-of-visual-thought: Teaching vlms to see and think better with continuous visual tokens. *arXiv preprint arXiv:2511.19418*, 2025.

Qwen Team. Qwen-image-edit-2511. `https://huggingface.co/Qwen/Qwen-Image-Edit-2511`, 2025. Hugging Face model card.

Qwen Team. Qwen3.8-Flash-Next: A new architecture, towards ultimate cost-efficiency, August 2026. URL `https://qwen.ai/blog?id=qwen3.8-flash-next`.

Andy Shih, Suneel Belkhale, Stefano Ermon, Dorsa Sadigh, and Nima Anari. Parallel sampling of diffusion models. *Advances in Neural Information Processing Systems*, 36:4263–4276, 2023.

Hannes Stärk, Bowen Jing, Regina Barzilay, and Tommi Jaakkola. Harmonic self-conditioned flow matching for multi-ligand docking and binding site design. *arXiv preprint arXiv:2310.05764*, 2023.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? a survey. *Transactions of the Association for Computational Linguistics*, 12:543–561, 2024.

Robin Strudel, Corentin Tallec, Florent Altché, Yilun Du, Yaroslav Ganin, Arthur Mensch, Will Grathwohl, Nikolay Savinov, Sander Dieleman, Laurent Sifre, et al. Self-conditioned embedding diffusion for text generation. *arXiv preprint arXiv:2211.04236*, 2022.

Alex Su, Haozhe Wang, Weiming Ren, Fangzhen Lin, and Wenhu Chen. Pixel reasoner: Incentivizing pixel space reasoning via curiosity-driven reinforcement learning. *Advances in Neural Information Processing Systems*, 38:8222–8251, 2026.

Zhaochen Su, Linjie Li, Mingyang Song, Yunzhuo Hao, Zhengyuan Yang, Jun Zhang, Guanjie Chen, Jiawei Gu, Juntao Li, Xiaoye Qu, et al. Openthinkimg: Learning to think with images via visual tool reinforcement learning. *arXiv preprint arXiv:2505.08617*, 2025a.

Zhaochen Su, Peng Xia, Hangyu Guo, Zhenhua Liu, Yan Ma, Xiaoye Qu, Jiaqi Liu, Yanshu Li, Kaide Zeng, Zhengyuan Yang, et al. Thinking with images for multimodal reasoning: Foundations, methods, and future frontiers. *arXiv preprint arXiv:2506.23918*, 2025b.

Jingqi Tong, Yurong Mou, Hangcheng Li, Mingzhe Li, Yongzhuo Yang, Ming Zhang, Qiguang Chen, Tianyi Liang, Xiaomeng Hu, Yining Zheng, et al. Thinking with video: Video generation as a promising multimodal reasoning paradigm. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 41121–41129, 2026.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Heribert Vollmer. *Introduction to circuit complexity: a uniform approach*. Springer Science & Business Media, 1999.

Qixun Wang, Yang Shi, Yifei Wang, Yuanxing Zhang, Pengfei Wan, Kun Gai, Xianghua Ying, and Yisen Wang. Monet: Reasoning in latent visual space beyond image and language. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12030–12040, 2026a.

Ruisi Wang, Zhongang Cai, Fanyi Pu, Junxiang Xu, Wanqi Yin, Maijunxian Wang, Ran Ji, Chenyang Gu, Bo Li, Ziqi Huang, et al. Demystifying video reasoning. *arXiv preprint arXiv:2603.16870*, 2026b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Thaddäus Wiedemer, Yuxuan Li, Paul Vicol, Shixiang Shane Gu, Nick Matarese, Kevin Swersky, Been Kim, Priyank Jaini, and Robert Geirhos. Video models are zero-shot learners and reasoners. *arXiv preprint arXiv:2509.20328*, 2025.

Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng-ming Yin, Shuai Bai, Xiao Xu, Yilei Chen, et al. Qwen-image technical report. *arXiv preprint arXiv:2508.02324*, 2025.

Zhen Xing, Qijun Feng, Haoran Chen, Qi Dai, Han Hu, Hang Xu, Zuxuan Wu, and Yu-Gang Jiang. A survey on video diffusion models. *ACM Computing Surveys*, 57(2):1–42, 2024.

Yi Xu, Chengzu Li, Han Zhou, Xingchen Wan, Caiqi Zhang, Anna Korhonen, and Ivan Vulić. Visual planning: Let's think only with images. In *International Conference on Learning Representations*, volume 2026, pp. 36706–36739, 2026.

Yao-Yuan Yang, Cyrus Rashtchian, Hongyang Zhang, Russ R Salakhutdinov, and Kamalika Chaudhuri. A closer look at accuracy vs. robustness. *Advances in neural information processing systems*, 33:8588–8601, 2020.

Zeyuan Yang, Xueyang Yu, Delin Chen, Maohao Shen, and Chuang Gan. Machine mental imagery: Empower multimodal reasoning with latent visual tokens. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 33510–33520, 2026.

Huanyu Zhang, Wenshan Wu, Chengzu Li, Ning Shang, Yan Xia, Yangyu Huang, Yifan Zhang, Li Dong, Zhang Zhang, Liang Wang, et al. Latent sketchpad: Sketching visual thoughts to elicit multimodal reasoning in mllms. *arXiv preprint arXiv:2510.24514*, 2025.

CONTENTS

1 Introduction **1**

2 Preliminaries **2**

2.1 Background . . . 3

2.2 Theoretical Setup and Notation . . . 3

3 Self-Conditioning Increases Expressive Power **4**

3.1 Vanilla Flow Matching Is Confined to Constant Depth . . . 4

3.2 Self-Conditioning . . . 5

3.3 Self-Conditioning Reaches PSPACE/poly . . . 6

3.4 Extensions . . . 6

4 Experiments **6**

4.1 Experimental Setup . . . 7

4.2 Empirical Validation of the Expressivity Separation . . . 7

4.3 Evaluation on Structured Visual-Reasoning Tasks . . . 8

4.4 Additional Empirical Analyses . . . 9

5 Related Work **9**

6 Conclusion and Discussion **9**

A Proofs of Main Expressivity Theorems **17**

A.1 Conventions and Preliminary Facts . . . 17

A.2 Expressiveness Upper Bound for VANILLAFM . . . 18

A.2.1 The Error Recurrence . . . 18

A.2.2 Proof of Lemma 1 . . . 19

A.2.3 Approximate Normalized Prefix Sums in AC^0 . . . 21

A.2.4 Proof of Theorem 1 . . . 21

A.3 Expressiveness Lower Bound for SELFCONDFM . . . 23

A.3.1 Local Simulation of Space-Bounded Machines . . . 23

A.3.2 Proof of Theorem 2 . . . 23

B Proofs of Theoretical Extensions **26**

B.1 The Growing Lipschitz Constant . . . 26

B.2 ℓ_p Lipschitz Condition . . . 26

B.3 Runge–Kutta Samplers . . . 28

B.4 Matching the Circuit Depth . . . 31

C Formalization and Circuit Complexity of the Tasks in §4.2 **33**

C.1 Task Descriptions 33
C.2 Preliminaries for the Complexity Analysis 33
C.3 Line Tracing 34
C.4 Long Multiplication 34
C.5 Long Text Rendering 35

D Experimental Details **38**
D.1 Overall Setup 38
D.2 Setup and Evaluation for the Tasks in §4.2 38
D.2.1 Multi-Seed Evaluation 39
D.2.2 Generated Examples 41
D.3 NFE Scaling 45
D.4 Setup and Results for Structured Visual-Reasoning Tasks 45
D.4.1 Jigsaw Results 45
D.4.2 Generated Examples 46
D.5 Additional Empirical Analyses 52
D.5.1 Empirical Lipschitz Estimation 52
D.5.2 Attention Allocation 52
D.5.3 Parallel and Revisable Reasoning 53

E Additional Related Work **55**
E.1 Generative Visual Reasoning 55
E.2 Computational Expressiveness 55

A Proofs of Main Expressivity Theorems

This appendix proves the two main theoretical results from §3: Theorem 1 and Theorem 2. Building on the theoretical setup in §2.2, Appendix A.1 first fixes the notation and the encoding in which circuits read and write elements of F_b, and collects the preliminary facts. Appendix A.2 then proves Theorem 1 with the help of Lemma 1, and Appendix A.3 proves Theorem 2. The extension results of §3.4 are proved separately in Appendix B.

A.1 Conventions and Preliminary Facts

Throughout this appendix, m denotes the problem size and $\mathsf{D} \in \{\mathsf{NC}^{k+1}, \mathsf{AC}^k, \mathsf{TC}^k\}$ denotes a circuit complexity class for a constant integer $k \geq 0$; we abbreviate $b = b(m)$, $n = n(m)$, $K = K(m)$, and $\delta := 2^{-b}$. All circuit families are non-uniform.

Relations among complexity classes. It is known that $\mathsf{NC}^k \subseteq \mathsf{AC}^k \subseteq \mathsf{TC}^k \subseteq \mathsf{NC}^{k+1}$ for all $k \geq 0$ (Arora et al., 2009). Consequently, $\mathsf{NC} = \bigcup_k \mathsf{AC}^k = \bigcup_k \mathsf{TC}^k$. Furthermore, $\mathsf{AC}^0 \subsetneq \mathsf{TC}^0$ because $\mathrm{PARITY} \in \mathsf{TC}^0 \setminus \mathsf{AC}^0$ (Furst et al., 1984; Hastad, 1986), whereas $\mathsf{TC}^0 \subsetneq \mathsf{NC}^1$ is widely conjectured but open (Vollmer, 1999). By definition, $\mathsf{P} \subseteq \mathsf{P/poly}$ and $\mathsf{PSPACE} \subseteq \mathsf{PSPACE/poly}$. Moreover, the class $\mathsf{P/poly}$ is exactly the class of languages decidable by polynomial-size families of Boolean circuits (Arora et al., 2009), so $\mathsf{NC} \subseteq \mathsf{P/poly} \subseteq \mathsf{PSPACE/poly}$. Both inclusions are widely conjectured to be strict (Arora et al., 2009).

Encoding of scalars. Fix a binary encoding of the scalars of F_b in which a scalar of magnitude at most 2^ℓ occupies $O(b + \ell)$ bits. Every scalar in the runs of this appendix has magnitude at most $\mathrm{poly}(m)$, i.e., $\ell = O(\log m)$, and all circuit constructions have $b = \mathrm{poly}(m)$. Their encodings thus have $\mathrm{poly}(m)$ bits. When b is additionally constrained to $O(\log m)$, the encodings have $O(\log m)$ bits.

Circuits for models. We say that a model u is computed by D circuits if, under the encoding above, its restriction to inputs of each total encoded length is computed by a D circuit of size polynomial in that length. The Boolean input w needs no convention of its own, since $\{0, 1\} \subset F_b$.

Finite-precision arithmetic. The grid $F_b = \{k2^{-b} : k \in \mathbb{Z}\}$ is closed under addition, under subtraction, and under multiplication by an integer, so these operations need no rounding. Iterated sums are independent of the association order. For every $x \in \mathbb{R}$ we have $|\,\mathrm{round}_b(x) - x| \leq \delta/2$, and round_b is nondecreasing and fixes F_b (i.e., $\mathrm{round}_b(x) = x$ for all $x \in F_b$).

Time grid. Both samplers (Algorithms 1 and 2) use discrete time steps $t_i = \mathrm{round}_b(s_i)$ with $s_i := i/K$; write $h_i := t_{i+1} - t_i$. Since round_b is nondecreasing and fixes 0 and 1, we have

$$t_0 = 0, \qquad t_K = 1, \qquad h_i \geq 0, \qquad \sum_{i=0}^{K-1} h_i = 1,$$

and $|t_i - s_i| \leq \delta/2$ gives $|h_i - 1/K| \leq \delta$.

Hardwiring the initialization. In both samplers the initial state $x_0 = \mathrm{round}_b(\tilde{x}_0)$ ranges over the finite set $(F_b \cap [-1, 1])^n$, since p_{prior} is supported on $[-1, 1]^n$, and the rest of the run is deterministic given x_0. If a model accepts $\mathcal{L}$ with margin c under a fixed sampler, the acceptance criterion holds for every x_0 occurring with positive probability. Since p_{prior} does not depend on w, fixing one such value $\hat{x}_0 = \hat{x}_0(m)$ for each m turns the sampler into a deterministic algorithm with the same guarantee. We do this whenever a sampler is regarded as a circuit family.

The table bound. A function f of few bits is shallow, however complicated it is: writing each output bit of an arbitrary $f : \{0, 1\}^j \to \{0, 1\}^{j'}$ as a disjunctive normal form over the j input literals gives an unbounded-fan-in circuit of depth two and size $O(j'2^j)$, hence of size $\mathrm{poly}(m)$ once $j = O(\log m)$ and $j' = \mathrm{poly}(m)$. We call this the *table bound*; it exploits no structure of f, only how many bits f reads and writes. In the proofs below, we frequently invoke it for maps on constantly many scalars, each of $O(\log m)$ bits.

A.2 Expressiveness Upper Bound for VanillaFM

The K Euler steps of VanillaFM are sequential, so a circuit that follows them has depth growing with K. We replace them by a Picard iteration (Hairer et al., 1993) that updates all K positions at once, and whose propagated error decays factorially across iteration rounds, up to an additive error floor (Appendix A.2.1). Therefore, $O(L)$ rounds already approximate the sampler's output to within a constant error; this proves Lemma 1 in Appendix A.2.2. Aside from the K parallel model calls, each round computes normalized prefix sums of the velocities, which Appendix A.2.3 approximates in AC^0 up to a constant error, small enough for acceptance with margin (see §2.2). Appendix A.2.4 substitutes these approximate sums into the Picard iteration to prove Theorem 1.

A.2.1 The Error Recurrence

Appendices A.2.2 and A.2.4 both compare an approximate trajectory with the exact one through the same error recurrence, so we bound it here once and for all.

Lemma 2. *Let $K \in \mathbb{N}^+$ be a positive integer, $B > 0$, $L \geq 1$, and $\alpha, \beta \geq 0$ be real numbers, and let $C := \lceil \mathrm{e}L + 4B \rceil$. Recall that $s_i := i/K$ (Appendix A.1). Let $(e_i^{(r)})_{0 \leq i \leq K,\, 0 \leq r \leq C}$ be nonnegative reals such that, for all i and all $r < C$,*

$$e_i^{(0)} \leq B s_i + \alpha, \tag{3}$$

$$e_i^{(r+1)} \leq \frac{L}{K} \sum_{j=0}^{i-1} e_j^{(r)} + \beta. \tag{4}$$

Then for all $0 \leq i \leq K$ and $0 \leq r \leq C$,

$$e_i^{(r)} \leq \frac{B L^r s_i^{r+1}}{(r+1)!} + \frac{\alpha L^r s_i^r}{r!} + \beta \sum_{q=0}^{r-1} \frac{L^q s_i^q}{q!}, \tag{5}$$

where $0^0 := 1$ and the sum is read as empty when $r = 0$. In particular, $e_K^{(C)} \leq \frac{1}{4} + \alpha + \beta \mathrm{e}^L$.

Proof. We prove (5) by induction on r. For $r = 0$ the right-hand side is $B s_i + \alpha$, so (5) is (3). Assume that (5) holds for r and all i. Since the function $\vartheta \mapsto \vartheta^q$ is nondecreasing on $[0, \infty)$ and $s_{j+1} - s_j = 1/K$, every integer $q \geq 0$ satisfies the Riemann bound

$$\frac{1}{K} \sum_{j=0}^{i-1} s_j^q = \frac{1}{K} \sum_{j=0}^{i-1} (j/K)^q \leq \int_0^{s_i} \vartheta^q \, \mathrm{d}\vartheta = \frac{s_i^{q+1}}{q+1}. \tag{6}$$

Summing (5) over $j < i$ and applying (6) termwise gives

$$\begin{aligned}
\frac{L}{K} \sum_{j=0}^{i-1} e_j^{(r)} &\leq \frac{1}{K} \sum_{j=0}^{i-1} \left(\frac{B L^{r+1} s_j^{r+1}}{(r+1)!} + \frac{\alpha L^{r+1} s_j^r}{r!} + \beta \sum_{q=0}^{r-1} \frac{L^{q+1} s_j^q}{q!} \right) \\
&\leq \frac{B L^{r+1} s_i^{r+2}}{(r+1)!\,(r+2)} + \frac{\alpha L^{r+1} s_i^{r+1}}{r!\,(r+1)} + \beta \sum_{q=0}^{r-1} \frac{L^{q+1} s_i^{q+1}}{q!\,(q+1)} \\
&= \frac{B L^{r+1} s_i^{r+2}}{(r+2)!} + \frac{\alpha L^{r+1} s_i^{r+1}}{(r+1)!} + \beta \sum_{q=1}^{r} \frac{L^q s_i^q}{q!}.
\end{aligned}$$

By (4), adding $\beta = \beta L^0 s_i^0 / 0!$ to this bound yields (5) for $r + 1$.

For the last claim, put $i = K$ and $r = C$ in (5), so that $s_K = 1$. Stirling's inequality $C! \geq (C/\mathrm{e})^C$ together with $C \geq \mathrm{e}L$ gives $L^C / C! \leq (\mathrm{e}L/C)^C \leq 1$, whence $\alpha L^C / C! \leq \alpha$ and

$$\frac{B L^C}{(C+1)!} = \frac{B}{C+1} \cdot \frac{L^C}{C!} \leq \frac{B}{4B+1} \leq \frac{1}{4}.$$

The remaining sum satisfies $\sum_{q<C} L^q/q! \leq \sum_{q \geq 0} L^q/q! = \mathrm{e}^L$, and adding the three terms yields

$$e_K^{(C)} \leq \frac{B L^C}{(C+1)!} + \frac{\alpha L^C}{C!} + \beta \sum_{q=0}^{C-1} \frac{L^q}{q!} \leq \frac{1}{4} + \alpha + \beta \mathrm{e}^L,$$

which completes the proof. □

A.2.2 PROOF OF LEMMA 1

Up to rounding, the K Euler updates of Algorithm 1 state that the trajectory $(x_i)_{i=0}^K$ solves the fixed-point system $x_i = x_0 + \sum_{j<i} h_j u(x_j, t_j; w)$, with $h_j := t_{j+1} - t_j$ as in Appendix A.1. Algorithm 3 solves this fixed-point system by a Picard iteration (Hairer et al., 1993): iteration r recomputes all positions $y_0^{(r+1)}, \dots, y_K^{(r+1)}$ in parallel from the previous iterate $(y_i^{(r)})_{i=0}^K$, replacing the rounded step sizes h_j by the uniform nominal step $1/K$, implemented by dividing each prefix sum $S_i^{(r)}$ by K (line 8 of Algorithm 3). We show that $C = O(L)$ iterations bring its endpoint $y_K^{(C)}$ within $\frac{1}{2}$ of x_K, thereby proving Lemma 1.

Algorithm 3 Parallel Picard Inference

1: **function** PARALLELPICARD($w; u, K, C$)
2: $t_i \leftarrow \mathrm{round}_b(i/K)$ for $i = 0, \dots, K$
3: Sample $\tilde{x}_0 \sim p_{\text{prior}}$; $x_0 \leftarrow \mathrm{round}_b(\tilde{x}_0)$
4: $y_i^{(0)} \leftarrow x_0$ for $i = 0, \dots, K$
5: **for** $r = 0, \dots, C-1$ **do**
6: **in parallel for** $j = 0, \dots, K-1$: $v_j^{(r)} \leftarrow u(y_j^{(r)}, t_j; w)$
7: $S_i^{(r)} \leftarrow \sum_{j=0}^{i-1} v_j^{(r)}$ for $i = 0, \dots, K$
8: **in parallel for** $i = 0, \dots, K$: $y_i^{(r+1)} \leftarrow \mathrm{round}_b\big(x_0 + S_i^{(r)}/K\big)$
9: **end for**
10: **return** $y_K^{(C)}$
11: **end function**

Proof of Lemma 1. Fix m and $w \in \{0,1\}^m$, and write $L = L(m)$. Let $c_0 = c_0(B) > 0$ be a constant to be chosen at the end of the proof, and take the inference algorithm to be PARALLELPICARD($w; u, K, C$) with

$$C := \lceil \mathrm{e}L + 4B \rceil.$$

Both VANILLAFM and PARALLELPICARD draw $\tilde{x}_0$ from p_{prior} and round it, so we may couple the two runs by giving them the same x_0. The estimates below hold for every realization of x_0, which is the only random quantity in either run, and therefore hold with probability 1.

The two trajectories. Let $(x_i)_{i=0}^K$ be the VANILLAFM trajectory and $v_j := u(x_j, t_j; w)$. In line 7 of Algorithm 1, the subtraction $t_{i+1} - t_i$ and the addition are exact in F_b (Appendix A.1). Only the product needs rounding, so $x_{i+1} = x_i + \mathrm{round}_b(h_i v_i)$, and each step contributes a rounding residual $\mathrm{round}_b(h_i v_i) - h_i v_i$ of size at most $\delta/2$. Summing these residuals into $\rho_i := \sum_{j<i} \big(\mathrm{round}_b(h_j v_j) - h_j v_j\big)$ gives

$$x_i = x_0 + \sum_{j=0}^{i-1} h_j v_j + \rho_i \qquad \left(\|\rho_i\|_\infty \le \frac{i\delta}{2} \le \frac{K\delta}{2} \right). \tag{7}$$

Likewise the prefix sum $S_i^{(r)}$ in line 7 of Algorithm 3 is a sum of elements of F_b and hence exact, and $x_0 \in F_b$, so line 8 rounds only once. Writing $\rho_i^{(r)}$ for this rounding residual, line 8 gives

$$y_i^{(r+1)} = x_0 + \frac{1}{K} \sum_{j=0}^{i-1} v_j^{(r)} + \rho_i^{(r)} \qquad \left(\|\rho_i^{(r)}\|_\infty \le \frac{\delta}{2} \right). \tag{8}$$

We bound the difference between (7) and (8) in two steps: we first verify the hypotheses (3) and (4) of Lemma 2 for it, and then apply the lemma. The two expressions differ in three respects, which the first step tracks: the weights $1/K$ against h_j, the velocities $v_j^{(r)}$ against v_j, and the rounding residuals ρ_i and $\rho_i^{(r)}$.

Verifying the hypotheses of Lemma 2. Put $e_i^{(r)} := \|y_i^{(r)} - x_i\|_\infty$, and set $\alpha := \frac{1}{2}(B+1)K\delta$ and $\beta := (B+1)K\delta$. For $r = 0$, using $h_j \geq 0$, $\sum_{j<i} h_j = t_i$, $t_i \leq s_i + \delta/2$ (Appendix A.1), and the B-boundedness $\|v_j\|_\infty \leq B$ of u, we have

$$e_i^{(0)} = \|x_0 - x_i\|_\infty \leq Bt_i + \frac{K\delta}{2} \leq Bs_i + \frac{(B+K)\delta}{2} \leq Bs_i + \alpha,$$

where the last step uses $B + K \leq (B+1)K$ for $K \geq 1$. This is (3) in Lemma 2. For $0 \leq r < C$, subtract (7) from (8):

$$y_i^{(r+1)} - x_i = \frac{1}{K}\sum_{j=0}^{i-1}\big(v_j^{(r)} - v_j\big) + \sum_{j=0}^{i-1}\Big(\frac{1}{K} - h_j\Big)v_j + \rho_i^{(r)} - \rho_i.$$

The two velocities in the first sum are evaluated at the same time t_j and the same condition w: $v_j^{(r)} = u(y_j^{(r)}, t_j; w)$ and $v_j = u(x_j, t_j; w)$. Therefore, L-Lipschitz continuity of u gives $\|v_j^{(r)} - v_j\|_\infty \leq Le_j^{(r)}$. By B-boundedness and $|h_j - 1/K| \leq \delta$ (Appendix A.1), the second sum is bounded by $B\sum_{j<K}|1/K - h_j| \leq BK\delta$. By the residual bounds in (7) and (8), the last two terms are bounded by $(K+1)\delta/2 \leq K\delta$. Thus,

$$e_i^{(r+1)} \leq \frac{L}{K}\sum_{j=0}^{i-1} e_j^{(r)} + BK\delta + K\delta = \frac{L}{K}\sum_{j=0}^{i-1} e_j^{(r)} + \beta,$$

which is (4) in Lemma 2.

Applying Lemma 2. With (3) and (4) verified, Lemma 2, applied with the parameters B, L, α, β above and the same $C = \lceil \mathrm{e}L + 4B \rceil$, yields $e_K^{(C)} \leq \frac{1}{4} + \alpha + \beta\mathrm{e}^L$. Since $\delta = 2^{-b}$, the precision hypothesis (2) is equivalent to $K\delta \leq 2^{-c_0}\mathrm{e}^{-L}$, so, using $\mathrm{e}^{-L} \leq 1$,

$$\alpha + \beta\mathrm{e}^L \leq \frac{(B+1)2^{-c_0}}{2} + (B+1)2^{-c_0} = \frac{3}{2}(B+1)2^{-c_0}.$$

Choose $c_0 = c_0(B)$ so that $\frac{3}{2}(B+1)2^{-c_0} \leq \frac{1}{4}$; then $\|y_K^{(C)} - x_K\|_\infty = e_K^{(C)} \leq \frac{1}{2}$.

Since u accepts $\mathcal{L}$ with margin 1 under VANILLAFM, every coordinate of x_K is at least 1 when $w \in \mathcal{L}$ and at most -1 otherwise, so every coordinate of the returned value $y_K^{(C)}$ is at least $\frac{1}{2}$ in the first case and at most $-\frac{1}{2}$ in the second. Thus u accepts $\mathcal{L}$ with margin $\frac{1}{2}$ under Algorithm 3, which runs for $C = O(L)$ iterations because $L \geq 1$ and B is a constant, and makes K parallel calls to u in each iteration. □

Aside from the model calls in line 6, one iteration of Algorithm 3 performs only the prefix sums of line 7 and the normalization of line 8. When $K = \mathrm{poly}(m)$ and $b = \mathrm{poly}(m)$, the prefix sums are iterated additions of polynomially many numbers of $\mathrm{poly}(m)$ bits each, while each normalization uses addition and division by the hardwired K, and TC^0 circuits compute all these operations exactly (Vollmer, 1999; Hesse et al., 2002). For constant L, Lemma 1 together with this observation essentially gives the TC^0 part of Theorem 1 (Appendix A.2.4 gives the complete proof for every D). The same iteration gives the growing-L extension in §3.4, i.e., Corollary 1.

Picard iterations have also been used to accelerate diffusion sampling (Shih et al., 2023), with a worst-case guarantee of as many iterations as sampling steps and fewer iterations observed empirically. We instead use Picard iteration as a proof device for expressivity: Lemma 1 shows that the K sequential steps of VANILLAFM collapse into $O(L)$ parallel rounds regardless of K, which caps the sequential depth that VANILLAFM can exploit.

As a side remark, this collapse relies on approximating the endpoint only to constant accuracy, which suffices for acceptance with margin. When exponentially small additive error is required, even a Lipschitz continuous ODE can be hard to solve: Kawamura (2010) constructs an ODE with a polynomial-time computable, Lipschitz continuous right-hand side whose solution is PSPACE-complete, with the answer encoded in the solution at precision $2^{-\mathrm{poly}(m)}$.

A.2.3 Approximate Normalized Prefix Sums in AC^0

The exact prefix sums in line 7 of Algorithm 3, however, lie beyond AC^0: the low-order bit of a sum is the parity of the summands' low-order bits (Furst et al., 1984; Hastad, 1986). To obtain the AC^0 part of Theorem 1, we approximate the sums, which suffices because the Picard iteration needs each normalized prefix sum only to within a constant additive error, which merely enlarges β in (4). Quantizing the summands to constantly many levels makes each normalized prefix sum an affine function of a *count of ones*, and AC^0 approximates such a count to within any constant fraction of the string length (Ajtai, 1990). Formally, we have the following lemma.

Lemma 3. *Let $M \geq 1$ and $\varepsilon \in (0,1)$ be constant reals. Let $N = N(m) \leq \mathrm{poly}(m)$ be a positive integer, and let the number of fractional bits $b = b(m) = O(\log m)$ satisfy $2^{-b} \leq \varepsilon$. There is a family of AC^0 circuits of size $\mathrm{poly}(m)$ that maps any input $g_0, \ldots, g_{N-1} \in F_b \cap [-M, M]$ to outputs $A_0, \ldots, A_N \in F_b$ with*

$$\Big| A_i - \frac{1}{N} \sum_{j=0}^{i-1} g_j \Big| \leq \varepsilon \qquad \textit{for all } 0 \leq i \leq N.$$

Proof. Fix a constant $R := \lceil 8M/\varepsilon \rceil$. For each j, let $k_j \in \{0, \ldots, R\}$ be the integer nearest to $(g_j + M)R/(2M)$ and put $\hat{g}_j := \frac{2M}{R} k_j - M$, so that $|\hat{g}_j - g_j| \leq M/R \leq \varepsilon/8$. Write $\chi_j \in \{0,1\}^R$ for the unary encoding of k_j. Since $g_j \in F_b \cap [-M, M]$ occupies $O(\log m)$ bits, the table bound (as in Appendix A.1) maps g_j to χ_j at depth two. For each i, the string $\chi_0 \cdots \chi_{i-1} 0^{(N-i)R}$ has length NR and exactly $S_i := \sum_{j<i} k_j$ ones. AC^0 circuits can count these ones up to any constant fraction $\gamma > 0$ of the length, i.e., output $\hat{S}_i$ with $|\hat{S}_i - S_i| \leq \gamma NR$ (Ajtai, 1990). Taking $\gamma := \varepsilon/(16M)$ gives $\frac{2M}{NR} |\hat{S}_i - S_i| \leq \varepsilon/8$. As $\frac{1}{N} \sum_{j<i} \hat{g}_j = \frac{2M}{NR} S_i - \frac{iM}{N}$, set

$$A_i := \mathrm{round}_b\Big(\frac{2M}{NR} \hat{S}_i - \frac{iM}{N} \Big).$$

Here i and N are hardwired and $\hat{S}_i$ occupies $O(\log m)$ bits, so the table bound maps $\hat{S}_i$ to A_i at depth two as well. Composing these three stages, each of constant depth and size $\mathrm{poly}(m)$, gives the claimed AC^0 family. It remains to bound the error of A_i. Inserting the intermediate quantities $\frac{2M}{NR} \hat{S}_i - \frac{iM}{N}$ and $\frac{1}{N} \sum_{j<i} \hat{g}_j$ and applying the triangle inequality,

$$\begin{aligned} \Big| A_i - \frac{1}{N} \sum_{j<i} g_j \Big| &\leq \Big| A_i - \Big(\frac{2M}{NR} \hat{S}_i - \frac{iM}{N} \Big) \Big| + \frac{2M}{NR} \big| \hat{S}_i - S_i \big| + \frac{1}{N} \sum_{j<i} |\hat{g}_j - g_j| \\ &\leq 2^{-b-1} + \frac{\varepsilon}{8} + \frac{\varepsilon}{8} \leq \frac{\varepsilon}{2} + \frac{\varepsilon}{4} \leq \varepsilon, \end{aligned}$$

the three terms being the rounding, counting, and quantization errors, and the second inequality using $2^{-b} \leq \varepsilon$. □

A.2.4 Proof of Theorem 1

To place the VANILLAFM sampler in D, we split on the number of steps: a bounded K is handled by unrolling the K Euler steps directly, and otherwise we substitute the approximate prefix sums of Lemma 3 into the Picard iteration of Lemma 1 (Algorithm 3). The substitution perturbs the error recurrence (4) by one constant per round, small enough for the endpoint to stay within a constant acceptance margin, and it keeps the circuit at constant depth over the model calls.

Proof of Theorem 1. By (1) there is a constant $c > 0$ such that $b \geq (1+c) \log_2 K$ for all but finitely many m. Let $c_1 = c_1(B) > 0$ satisfy

$$\Big(\frac{3B}{2} + 1 \Big) 2^{-c_1} \leq \frac{1}{8}, \tag{9}$$

and split the proof into the two cases $b < \log_2 K + (\log_2 \mathrm{e}) L + c_1$ and $b \geq \log_2 K + (\log_2 \mathrm{e}) L + c_1$. Since the circuit families are non-uniform, each m may use the construction of its own case.

Small K. Suppose first that $b < \log_2 K + (\log_2 \mathrm{e})L + c_1$. Then K is bounded: only finitely many such m have $b < (1+c)\log_2 K$ by (1), and every other one satisfies $(1+c)\log_2 K \le b < \log_2 K + (\log_2 \mathrm{e})L + c_1$, hence $K < 2^{((\log_2 \mathrm{e})L + c_1)/c}$, a constant.

We then reproduce the sampler directly. Fix m and a hardwired initialization $\hat{x}_0$ as in Appendix A.1. The circuit runs the K Euler steps of Algorithm 1 from $x_0 := \hat{x}_0$ with the time grid hardwired: step i evaluates $v_i := u(x_i, t_i; w)$ with one copy of u and then sets $x_{i+1} := x_i + \mathrm{round}_b(h_i v_i)$ coordinate-wise, where $h_i := t_{i+1} - t_i$. This reproduces the VANILLAFM trajectory exactly, so its endpoint has every coordinate at least 1 when $w \in \mathcal{L}$ and at most -1 otherwise, and the sign of the first coordinate decides $\mathcal{L}$.

The circuit lies in D. Each of the constantly many steps makes one call to the D circuit u. Each coordinate-wise update is a map on constantly many scalars of $O(\log m)$ bits, since $b = O(\log m)$ and, with K bounded in this case, every state and velocity has magnitude bounded by a constant. The table bound places this map in $\mathsf{AC}^0 \subseteq \mathsf{D}$ (Appendix A.1). Composing constantly many D circuits keeps the result in D, so $\mathcal{L} \in \mathsf{D}$ in this case.

Large K. Otherwise $b \ge \log_2 K + (\log_2 \mathrm{e})L + c_1$. We construct the circuit by running Algorithm 3 with its exact prefix sums replaced by the approximate ones of Lemma 3, and then show that it decides $\mathcal{L}$ and lies in D. Set $M := \max\{B, 1\}$ and $\eta := \frac{1}{8}\mathrm{e}^{-L}$, and use the same $C = \lceil \mathrm{e}L + 4B \rceil$ as in Appendix A.2.2. This lower bound on b is equivalent to $K\delta \le 2^{-c_1}\mathrm{e}^{-L}$, which with (9) gives $\delta \le K\delta \le \eta$ and, by (1), $K \le 2^b = \mathrm{poly}(m)$.

The circuit. Fix m and a hardwired initialization $\hat{x}_0$ as in Appendix A.1, and let Φ_m be the circuit of Lemma 3 for $N := K$, M, and $\varepsilon := \eta$, whose hypotheses hold by the previous paragraph and (1). On input w, the circuit runs C rounds of Algorithm 3 from $y_i^{(0)} := \hat{x}_0$, with the time grid hardwired and lines 7 and 8 replaced by $y_i^{(r+1)} := \hat{x}_0 + A_i^{(r)}$. Here $A_0^{(r)}, \dots, A_K^{(r)}$ are the outputs of Φ_m applied coordinate-wise to the velocities $v_j^{(r)} := u(y_j^{(r)}, t_j; w)$, $j = 0, \dots, K-1$. Each $A_i^{(r)}$ therefore approximates the normalized prefix sum $\frac{1}{K}\sum_{j<i} v_j^{(r)}$. B-boundedness puts the velocities in $F_b \cap [-M, M]$ as Lemma 3 requires. Their normalized prefix sums also lie in $[-M, M]$, so every scalar here has constant magnitude and hence $O(\log m)$ bits, as the table bound below requires.

Correctness. We show that the circuit decides $\mathcal{L}$ by the sign of $(y_K^{(C)})_1$. Compare with the VANILLAFM run from $x_0 = \hat{x}_0$, with v_j and ρ_i as in (7), and put $e_i^{(r)} := \|y_i^{(r)} - x_i\|_\infty$. Since $\hat{x}_0, A_i^{(r)} \in F_b$, the new update involves no rounding, so subtracting (7) from $y_i^{(r+1)} = \hat{x}_0 + A_i^{(r)}$ gives

$$y_i^{(r+1)} - x_i = \Big(A_i^{(r)} - \tfrac{1}{K}\textstyle\sum_{j<i} v_j^{(r)}\Big) + \tfrac{1}{K}\textstyle\sum_{j<i}(v_j^{(r)} - v_j) + \sum_{j<i}\Big(\tfrac{1}{K} - h_j\Big)v_j - \rho_i.$$

The first term is bounded by η by Lemma 3; the remaining three terms are bounded by $\frac{L}{K}\sum_{j<i} e_j^{(r)}$, $BK\delta$, and $K\delta/2$ exactly as in Appendix A.2.2, with $\rho_i^{(r)}$ absent. Thus (3) holds with the same $\alpha := \frac{1}{2}(B+1)K\delta$ and (4) with $\beta := \eta + (B + \frac{1}{2})K\delta$, and Lemma 2 gives

$$e_K^{(C)} \le \frac{1}{4} + \alpha + \beta\mathrm{e}^L = \frac{1}{4} + \eta\mathrm{e}^L + \Big(\frac{B+1}{2} + \Big(B + \frac{1}{2}\Big)\mathrm{e}^L\Big)K\delta \le \frac{1}{4} + \frac{1}{8} + \Big(\frac{3B}{2} + 1\Big)2^{-c_1} \le \frac{1}{2},$$

using $\eta\mathrm{e}^L = \frac{1}{8}$, then $K\delta\mathrm{e}^L \le 2^{-c_1}$ and $K\delta \le 2^{-c_1}$, and finally (9). As in Appendix A.2.2, acceptance with margin 1 under VANILLAFM at the positive-probability initialization $\hat{x}_0$ (Appendix A.1) now gives $(y_K^{(C)})_1 \ge \frac{1}{2}$ if $w \in \mathcal{L}$ and $(y_K^{(C)})_1 \le -\frac{1}{2}$ otherwise.

Size and depth. The run consists of C rounds, each consisting of K parallel calls to u surrounded by AC^0 circuits of size $\mathrm{poly}(m)$, namely n copies of Φ_m, one per state coordinate, and, by the table bound, the addition of $\hat{x}_0$. Since $C = O(1)$ and K, n are $\mathrm{poly}(m)$, the whole circuit has size $\mathrm{poly}(m)$ and depth $O(1)$ over the copies of u. The surrounding arithmetic lies in $\mathsf{AC}^0 \subseteq \mathsf{D}$, and composing these C layers of parallel copies of the D circuit u keeps the result in D. Since the sign test $(y_K^{(C)})_1 \ge 0$ reads only the $O(\log m)$ bits of one scalar, the table bound computes it in AC^0, correctly outputting $\mathbf{1}\{w \in \mathcal{L}\}$. Hence $\mathcal{L} \in \mathsf{D}$. □

A.3 EXPRESSIVENESS LOWER BOUND FOR SELFCONDFM

We now prove Theorem 2 by making SELFCONDFM simulate a Turing machine, using the endpoint prediction z_i as the Turing machine's tape. Appendix A.3.1 puts a space-bounded Turing machine with advice into a form in which the new value of each cell is determined by that cell and its two neighbors. Appendix A.3.2 then builds a model that applies this rewrite rule to z_i coordinate-wise, so that each output coordinate reads constantly many inputs and the model stays in AC^0. Since the extrapolation $z_{i+1} = x_i + (1 - t_i)v_i$ (line 8 of Algorithm 2) carries the current velocity with sufficient magnitude while t_i is small, the tape can advance one machine step per sampling step, and taking K large enough lets the simulated Turing machine halt. The tape then freezes on the verdict, which keeps the velocity constant and decides the sign of x_K.

A.3.1 LOCAL SIMULATION OF SPACE-BOUNDED MACHINES

To prepare the machine for the coordinate-wise simulation of Appendix A.3.2, we flatten it into a cellular automaton: the initial tape is the input followed by one hardwired string carrying the advice, and the machine fills the tape with one of two verdict symbols before halting.

Lemma 4. *Let $\mathcal{L}$ be any language in* $\mathsf{PSPACE/poly}$. *There exist an alphabet $\Sigma \supseteq \{0,1\}$ of constant size containing two symbols* acc *and* rej, *a boundary symbol $\# \notin \Sigma$, a local rule $\varphi : (\Sigma \cup \{\#\})^3 \to \Sigma$, a tape length $P(m) = \mathrm{poly}(m)$, a horizon $H(m) \le 2^{\mathrm{poly}(m)}$, and strings $\hat{a}_m \in \Sigma^{P(m)-m}$, such that the following holds for every $w \in \{0,1\}^m$. The sequence*

$$c^{(1)} := \mathrm{concat}(w, \hat{a}_m), \qquad c_p^{(i+1)} := \varphi\big(c_{p-1}^{(i)}, c_p^{(i)}, c_{p+1}^{(i)}\big) \quad (1 \le p \le P(m)),$$

with the convention $c_0^{(i)} = c_{P(m)+1}^{(i)} = \#$, satisfies $c^{(i)} = \mathsf{acc}^{P(m)}$ for all $i \ge H(m)$ if $w \in \mathcal{L}$, and $c^{(i)} = \mathsf{rej}^{P(m)}$ otherwise. If moreover $\mathcal{L} \in \mathsf{P/poly}$, then $H(m) = \mathrm{poly}(m)$.

Proof. Take a deterministic Turing machine $\mathcal{M}$ that decides $w \in \mathcal{L}$ on the cells $1, \dots, P(m)$, for some $P(m) = \mathrm{poly}(m)$, from the tape initialization $\mathrm{concat}(w, \diamond, a_m)$, with $\diamond$ a separator and a_m an advice string of length $P(m) - m - 1$. Being deterministic and halting, $\mathcal{M}$ never repeats one of its at most $2^{O(P(m))}$ configurations, so it halts within $T(m) \le 2^{\mathrm{poly}(m)}$ steps, and within $T(m) \le \mathrm{poly}(m)$ steps in the $\mathsf{P/poly}$ case, where $\mathcal{M}$ can be taken polynomial-time. We may assume that $\mathcal{M}$ starts in an initial state q_0 with its head on the separator and, before halting, overwrites every cell with acc or rej according to its verdict.

Fold the head and its state into the alphabet: $\Sigma := \Gamma \cup (\mathcal{Q} \times \Gamma)$, where $\Gamma \supseteq \{0, 1, \diamond, \mathsf{acc}, \mathsf{rej}\}$ is the tape alphabet and $\mathcal{Q}$ the state set. The whole run of $\mathcal{M}$ is then local: each cell's next content is determined by itself and its two neighboring cells. We encode this one-step update by the rule φ, with $\#$ read as an end-marker and a halting head dissolving into its verdict. Since a cell changes only when the head acts on it, the headless all-acc and all-rej configurations are fixed points. Taking $\hat{a}_m := \mathrm{concat}((q_0, \diamond), a_m)$ makes $c^{(1)}$ the initial configuration, so $c^{(i)}$ is the configuration after $i-1$ steps. A halting head thus appears by $c^{(T(m)+1)}$ and dissolves one step later, leaving the tape filled with the verdict, a fixed point. Hence $H(m) := T(m) + 2$ works. □

A.3.2 PROOF OF THEOREM 2

To make the sampler simulate the machine of Lemma 4, we store the binary encoding of $c^{(i)}$ in the prediction z_i, one bit per coordinate, and take $K = 16H$. The simulation then runs at times $t_i \le \frac{1}{8}$, where the extrapolation $z_{i+1} = x_i + (1 - t_i)v_i$ delivers the velocity with sufficient magnitude, so z_{i+1} decodes exactly to the next configuration computed by the model. Once the tape freezes on the verdict, the velocity stays pinned at ± 4, ensuring that every coordinate of x_K clears the acceptance margin with the sign of the verdict.

Proof of Theorem 2. Let Σ, φ, $P = P(m)$, $H = H(m)$, and $\hat{a}_m$ be as in Lemma 4. Put $\kappa := \lceil \log_2 |\Sigma| \rceil$ and fix an injection $\mathrm{enc} : \Sigma \to \{0,1\}^\kappa$ with $\mathrm{enc}(\mathsf{acc}) = \mathbf{1}^\kappa$ and $\mathrm{enc}(\mathsf{rej}) = \mathbf{0}^\kappa$, calling $\mathrm{enc}(\sigma)$ the *codeword* of σ. Extend enc to configurations $c \in \Sigma^P$ cell by cell, $\mathrm{enc}(c) := \mathrm{concat}(\mathrm{enc}(c_1), \dots, \mathrm{enc}(c_P))$, and set

$$n := \kappa P, \qquad K := 16H.$$

The codeword of cell p thus occupies the *block* of coordinates $\kappa(p-1)+1,\dots,\kappa p$, so, with $p := \lceil j/\kappa \rceil$, coordinate j carries bit $q := j-\kappa(p-1)$ of the codeword of cell p. Let $N_p := \{p-1, p, p+1\} \cap \{1,\dots,P\}$ and let $J_j := \bigcup_{r\in N_p}\{\kappa(r-1)+1,\dots,\kappa r\}$ collect the blocks of cell p and of its neighbors on the tape, and let $I_j := \{p\}$ if $p \le m$ and $I_j := \emptyset$ otherwise. The precision $b = b(m)$ is fixed at the end of the proof. Everything below holds for any choice with $b \ge 3$ and $b = O(\log m)$. The next three paragraphs construct u and verify conditions (a) and (b) of Theorem 2, the paragraph *The run* proves acceptance, and the paragraph *Parameters* fixes b to meet condition (c).

Encoding a bit. A coordinate of the prediction z_i encodes one bit of a configuration, or the placeholder $\perp$ when not yet initialized, via the *encoding set* E and the decoding map dec given by

$$E := \{0\} \cup \{\zeta \in F_b : 2 \le |\zeta| \le 5\},$$
$$\mathrm{dec}(\zeta) := \begin{cases} \perp & \zeta = 0, \\ 1 & \zeta \ge 2, \\ 0 & \zeta \le -2. \end{cases}$$

The three bands $\{0\}$, $E\cap[2,5]$, and $E\cap[-5,-2]$ are deliberately wide and pairwise at distance at least 2. Therefore, the encoding can tolerate the sampler's perturbation, while an 8-Lipschitz model can still tell the bands apart.

The local maps. Define $\tau_j : \{0,1\}^{I_j} \times \{0,1,\perp\}^{J_j} \to \{0,1\}$ by

$$\tau_j(\omega,\psi) := \begin{cases} \text{bit } q \text{ of } \mathrm{enc}(\omega_p) & \psi = \perp^{J_j} \text{ and } p \le m, \\ \text{bit } q \text{ of } \mathrm{enc}\big((\hat a_m)_{p-m}\big) & \psi = \perp^{J_j} \text{ and } p > m, \\ \text{bit } q \text{ of } \mathrm{enc}\big(\varphi(\sigma_{p-1},\sigma_p,\sigma_{p+1})\big) & \psi = (\mathrm{enc}(\sigma_r))_{r\in N_p}, \\ 0 & \text{otherwise,} \end{cases}$$

with the convention $\sigma_0 = \sigma_{P+1} = \#$ at the tape ends. The first two cases initialize the tape and the third advances the machine by one step, so that with $\bar c^{(0)} := \perp^n$ and $\bar c^{(i)} := \mathrm{enc}(c^{(i)})$ for $i \ge 1$,

$$\bar c_j^{(i+1)} = \tau_j\big(w_{I_j}, \bar c_{J_j}^{(i)}\big) \qquad \text{for all } j \text{ and all } i \ge 0. \tag{10}$$

The model. Write $D_j := \{0,1\}^{I_j} \times E^{J_j}$, a finite nonempty subset of $F_b^{I_j} \times F_b^{J_j}$, and define $f_j : D_j \to \{-4,4\}$ by

$$f_j(\omega,\psi) := \begin{cases} +4 & \tau_j\big(\omega, \mathrm{dec}(\psi)\big) = 1, \\ -4 & \tau_j\big(\omega, \mathrm{dec}(\psi)\big) = 0, \end{cases}$$

where dec is applied coordinate-wise to ψ. Then f_j is 8-Lipschitz on D_j in the maximum norm: if two points have different values, they differ in a Boolean coordinate or lie in different bands at some E coordinate, so they differ by at least 1 in that coordinate, while the values differ by 8.

Extend f_j to all of $F_b^{I_j} \times F_b^{J_j}$ by a clipped McShane extension (McShane, 1934),

$$\bar f_j(\omega,\psi) := \max\Big\{-4,\ \min\Big\{4,\ \min_{(\omega',\psi')\in D_j}\big(f_j(\omega',\psi') + 8\max\{\|\omega-\omega'\|_\infty, \|\psi-\psi'\|_\infty\}\big)\Big\}\Big\}.$$

Then $\bar f_j$ is 8-Lipschitz and 4-bounded, and agrees with f_j on D_j. It needs no rounding, because F_b is closed under the arithmetic involved.

For every m, define the restriction of the single model u with $b = b(m)$, $n = n(m)$, and $|\mathrm{ctx}| = m + n$ by

$$\big(u(x,t;\mathrm{ctx})\big)_j := \bar f_j\big(\mathrm{ctx}_{I_j}, (\mathrm{ctx}_{m+k})_{k\in J_j}\big), \qquad j = 1,\dots,n,$$

and set $u(x,t;\mathrm{ctx}) := \mathbf{0}^n$ on all other encoded inputs. Since each output coordinate is $\bar f_j$ of a fixed subtuple of the condition, the model is 4-bounded and 8-Lipschitz continuous in the sense of §2.2. Each $\bar f_j$ reads $|I_j| + |J_j| = O(1)$ scalars, and an AC^0 circuit first clips them to $[-8,8]$. This leaves $\bar f_j$ unchanged, since $D_j \subseteq [-5,5]^{I_j\cup J_j}$ and a scalar of magnitude at least 8 forces $\bar f_j = 4$. The clipped scalars have $O(\log m)$ bits, so the table bound (Appendix A.1) computes $\bar f_j$ at depth two and size $\mathrm{poly}(m)$. Hence u is computed by AC^0 circuits, and conditions (a) and (b) of Theorem 2 hold.

The run. We now prove that u accepts $\mathcal{L}$ with margin 1 under SELFCONDFM. Fix $w \in \{0,1\}^m$ and run SELFCONDFM$(w; u, K)$. The estimates below hold for every realization of x_0, which is the only randomness in the run, and therefore hold with probability 1. By line 6 of Algorithm 2, its condition at step i is $\mathrm{concat}(w, z_i)$, of length exactly $m+n$, so $(v_i)_j = \bar{f}_j(w_{I_j}, (z_i)_{J_j})$. The run splits at step H. For the first phase, when $0 \le i \le H$, one sampling step advances the tape by one machine step, so that z_i decodes to $\bar{c}^{(i)} = \mathrm{enc}(c^{(i)})$. For the second phase, from H on, Lemma 4 freezes the tape on the verdict, the velocity stays pinned at ± 4, and x_K inherits this sign. We prove these two claims in turn.

For the first phase, we show that z_i decodes to $\bar{c}^{(i)}$ for every $0 \le i \le H$. This phase runs at small times: since $b \ge 3$, we have $\delta/2 \le 2^{-4}$, so $t_i \le i/K + 2^{-4}$, and $K = 16H$ gives

$$t_i \le \frac{1}{16} + \frac{1}{16} = \frac{1}{8} \qquad \text{for all } 0 \le i \le H. \tag{11}$$

We prove by induction on i the stronger claim that for every $0 \le i \le H$: (a) $(z_i)_j \in E$ with $\mathrm{dec}((z_i)_j) = \bar{c}^{(i)}_j$ for all j; (b) $\|x_i\|_\infty \le 1 + 4t_i$; and (c) all arithmetic up to step i is exact. All three hold at $i = 0$. Given them at some $i < H$, claim (a) puts the argument of $\bar{f}_j$ in D_j, where $\bar{f}_j$ agrees with f_j, so that $(v_i)_j$ is $+4$ if $\bar{c}^{(i+1)}_j = 1$ and -4 otherwise, by (10). As F_b is closed under multiplication by integers, both updates in lines 7 and 8 of Algorithm 2 are exact, giving (c) and $\|x_{i+1}\|_\infty \le \|x_i\|_\infty + 4(t_{i+1} - t_i) \le 1 + 4t_{i+1}$, which is (b). For (a), if $\bar{c}^{(i+1)}_j = 1$ then $(z_{i+1})_j = (x_i)_j + 4(1 - t_i)$ lies in $[3 - 8t_i,\, 5] \subseteq [2, 5]$ by (b) and (11), and symmetrically in $[-5, -2]$ if $\bar{c}^{(i+1)}_j = 0$. This completes the induction, so z_H decodes to $\bar{c}^{(H)}$, the encoded verdict tape.

For the second phase, we show that the velocity stays at ± 4 from step H on, so that every coordinate of x_K has the sign of the verdict and magnitude at least 1. The first-phase bound on z_{i+1} relied on $t_i \le 1/8$, which fails as t_i approaches 1, so we use instead that the tape is frozen. Suppose that $w \in \mathcal{L}$. Then Lemma 4 and $\mathrm{enc}(\mathsf{acc}) = \mathbf{1}^\kappa$ give $\bar{c}^{(i)} = \mathbf{1}^n$ for all $i \ge H$. We show by induction on $H \le i \le K$ that (a′) $(z_i)_j \in E \cap [2, 5]$ for all j, and (b′) $x_i = x_H + 4(t_i - t_H)\mathbf{1}^n$ exactly. At $i = H$, (a′) is (a) with $\bar{c}^{(H)} = \mathbf{1}^n$, and (b′) is trivial. Given them at some $i < K$, (a′) puts the argument of $\bar{f}_j$ in D_j with $\mathrm{dec}((z_i)_{J_j}) = \mathbf{1}^{J_j} = \bar{c}^{(i)}_{J_j}$, so (10) and $\bar{c}^{(i+1)} = \mathbf{1}^n$ give $(v_i)_j = +4$ for every j. Line 7 of Algorithm 2 then gives (b′) at $i + 1$, and in line 8 the two t_i terms cancel:

$$(z_{i+1})_j = (x_i)_j + 4(1 - t_i) = (x_H)_j + 4(1 - t_H) \in [3 - 8t_H,\, 5] \subseteq [2, 5]$$

by (b) and (11) at $i = H$, which is (a′) at $i + 1$. Thus z_i stops moving and the velocity stays at $+4$ until the end. Taking $i = K$ in (b′) and using $t_K = 1$ gives $(x_K)_j = (x_H)_j + 4(1 - t_H) \ge 2 \ge 1$ for every j. Symmetrically, if $w \notin \mathcal{L}$, then $\bar{c}^{(i)} = \mathbf{0}^n$ for $i \ge H$, the velocity stays at -4, and $(x_K)_j \le -1$. Hence u accepts $\mathcal{L}$ with margin 1 under SELFCONDFM.

Parameters. Finally, we fix b and read off the two cases of Theorem 2. For $\mathcal{L} \in \mathsf{PSPACE/poly}$, Lemma 4 gives $n(m) = \kappa P(m) = \mathrm{poly}(m)$ and $K(m) = 16H(m) \le 2^{\mathrm{poly}(m)}$, and $b(m) := 3 = O(\log m)$ meets the sole requirement $b \ge 3$ made above. If moreover $\mathcal{L} \in \mathsf{P/poly}$, the same lemma gives $H(m) = \mathrm{poly}(m)$ and hence $K(m) = \mathrm{poly}(m)$, while b must in addition satisfy the bound (1) on its fractional bits. The choice

$$b(m) := \max\{3,\ \lceil 2 \log_2 K(m) \rceil\}$$

satisfies $b(m) \ge 2 \log_2 K(m)$, and hence (1). Since $K(m)$ is now polynomial, this $b(m)$ is $O(\log m)$, and the construction remains valid. In both cases condition (c) holds, which completes the proof. □

B PROOFS OF THEORETICAL EXTENSIONS

This appendix proves the extension results presented in §3.4. Appendix B.1 establishes Corollary 1, the circuit analogue of Lemma 1 for a growing Lipschitz constant L. Appendix B.2 extends the analyses from ℓ_∞ to ℓ_p Lipschitz continuity for any fixed $p \in [1, \infty]$. Appendix B.3 extends the upper bounds for VANILLAFM from the Euler method to fixed explicit Runge–Kutta samplers. Finally, Appendix B.4 encodes majority circuits into VANILLAFM, demonstrating that the $O(L)$ rounds of Lemma 1 match circuit depth and cannot be improved without collapsing the majority-circuit depth hierarchy.

B.1 THE GROWING LIPSCHITZ CONSTANT

The proof of Lemma 1 bounds the number of Picard rounds by $O(L)$ without using the constancy of L, so the lemma already allows a growing $L(m)$. Since the exact prefix sums of each round are computable in TC^0 when $K = \mathrm{poly}(m)$ and $b = \mathrm{poly}(m)$, the whole run then becomes a majority circuit of depth $O(L)$ instead of $O(1)$.

Corollary 1. *Let u be a model that can be computed by circuits in TC^0, and $\mathcal{L}$ be any language that can be accepted by u with margin 1 under* VANILLAFM. *Assume that u is B-bounded for a constant $B > 0$ and $L(m)$-Lipschitz continuous for some $L(m) \geq 1$ that may grow with m. Then there exists a constant $c_0 = c_0(B) > 0$, independent of m, such that as long as the state dimension and the number of steps satisfy $n = \mathrm{poly}(m)$ and $K = \mathrm{poly}(m)$, and the number of fractional bits satisfies*

$$\log_2 K(m) + (\log_2 \mathrm{e})L(m) + c_0 \leq b(m) \leq \mathrm{poly}(m) \qquad \textit{for every } m, \tag{12}$$

$\mathcal{L}$ can be decided by majority circuits of size $\mathrm{poly}(m)$ and depth $O(L(m))$. In particular, $\mathcal{L} \in \mathsf{TC}^k$ if $L(m) = O(\log^k m)$ for a constant integer $k \geq 0$, and $\mathcal{L} \in \mathsf{NC}$ if $L(m) = \mathrm{poly}(\log m)$.

Proof. We turn the algorithm of Lemma 1 into a majority circuit. The lower bound in (12) is exactly the hypothesis (2) of Lemma 1, with $c_0 = c_0(B)$ the constant of that lemma. Hence the lemma's algorithm, namely Algorithm 3 with $C := \lceil \mathrm{e}L + 4B \rceil$ rounds, run from a hardwired initialization $\hat{x}_0$ (Appendix A.1), accepts $\mathcal{L}$ with margin $\frac{1}{2}$: every coordinate of its output $y_K^{(C)}$ is at least $\frac{1}{2}$ when $w \in \mathcal{L}$ and at most $-\frac{1}{2}$ otherwise.

As noted after the proof of Lemma 1, one round of that algorithm consists of K parallel calls to u followed by prefix sums and normalizations, which TC^0 computes exactly since K and b are $\mathrm{poly}(m)$ (Vollmer, 1999; Hesse et al., 2002). Since u is itself computed by TC^0 circuits, composing the C rounds with the indicator $\mathbf{1}\{(y_K^{(C)})_1 \geq 0\}$ yields a majority circuit deciding $\mathcal{L}$. Its depth is $O(C) = O(L(m))$, and its size is polynomial because each round has size $\mathrm{poly}(m)$ and the precision hypothesis (12) bounds $(\log_2 \mathrm{e})L(m)$ by $b(m) = \mathrm{poly}(m)$, so that $C = \mathrm{poly}(m)$.

If $L(m) = O(\log^k m)$, the depth is $O(\log^k m)$ and $\mathcal{L} \in \mathsf{TC}^k$ by definition. If $L(m) = \mathrm{poly}(\log m)$, then $L(m) = O(\log^k m)$ for some constant k, so $\mathcal{L} \in \mathsf{TC}^k \subseteq \mathsf{NC}$ (Appendix A.1). □

B.2 ℓ_p LIPSCHITZ CONDITION

We now extend Theorems 1 and 2 to the ℓ_p norm for a fixed $p \in [1, \infty]$. We say that u is *L-Lipschitz in ℓ_p* if

$$\|u(x, t; \mathrm{ctx}) - u(x', t; \mathrm{ctx}')\|_p \leq L \max\{\|x - x'\|_p, \|\mathrm{ctx} - \mathrm{ctx}'\|_p\}$$

for all $x, x' \in F_b^n$, equal-length $\mathrm{ctx}, \mathrm{ctx}' \in F_b^*$, and $t \in [0, 1] \cap F_b$.

Proposition 1 (Upper bound under ℓ_p)**.** *Theorem 1 remains true if its ℓ_∞ Lipschitz hypothesis is replaced by an ℓ_p Lipschitz hypothesis for any fixed $p \in [1, \infty]$.*

Proof. The small-K case of Appendix A.2.4 re-runs the sampler exactly and is independent of the Lipschitz norm, so it carries over unchanged. For the large-K case, we follow the proof of Theorem 1 in Appendix A.2.4, measuring every error in the normalized norm $e_i^{(r)} := n^{-1/p}\|y_i^{(r)} - x_i\|_p$, with $n^{-1/\infty} := 1$. Two observations carry the argument over. Boundedness gives $\|v_j\|_p \leq$

$n^{1/p}B$, and a perturbation bounded by ε in every coordinate changes the normalized norm by at most ε, both because $\|\cdot\|_p \le n^{1/p}\|\cdot\|_\infty$. The ℓ_p Lipschitz hypothesis gives $\|v_j^{(r)} - v_j\|_p \le L\|y_j^{(r)} - x_j\|_p$, since the two velocities share t_j and w.

We run the construction of Appendix A.2.4 with $\eta := \frac{1}{16}\mathrm{e}^{-L}$ in place of $\frac{1}{8}\mathrm{e}^{-L}$, choosing the constant $c_1 = c_1(B)$ of (9) large enough that $(\frac{3B}{2}+1)2^{-c_1} \le \frac{1}{16}$ and $2^{-c_1} \le \frac{1}{16}$. In the large-$K$ case we have $b \ge \log_2 K + (\log_2 \mathrm{e})L + c_1$, which is equivalent to $K\delta \le 2^{-c_1}\mathrm{e}^{-L}$ and thus gives $\delta \le K\delta \le \eta$, so Lemma 3 applies with $\varepsilon := \eta$ exactly as there, and every other parameter is unchanged.

The error analysis of Appendix A.2.4 then carries over: with $s_i := i/K$ and the same $\alpha := \frac{B+1}{2}K\delta$ and $\beta := \eta + (B + \frac{1}{2})K\delta$ as there,

$$e_i^{(0)} \le Bs_i + \alpha, \qquad e_i^{(r+1)} \le \frac{L}{K}\sum_{j=0}^{i-1} e_j^{(r)} + \beta.$$

Indeed, the four terms of the decomposition of Appendix A.2.4 are bounded exactly as there: the prefix-sum error contributes η, the velocity differences $\frac{L}{K}\sum_{j<i} e_j^{(r)}$ by the second observation, the weight mismatch at most $BK\delta$ by the first, and the rounding residuals $K\delta/2$ since $n^{-1/p}\|\cdot\|_p \le \|\cdot\|_\infty$, and similarly for the initial bound.

Lemma 2 with $C := \lceil \mathrm{e}L + 4B \rceil$ gives $e_K^{(C)} \le \frac{1}{4} + \alpha + \beta\mathrm{e}^L$, and as in Appendix A.2.4 the precision hypothesis bounds $\alpha + \beta\mathrm{e}^L \le (\frac{3B}{2}+1)2^{-c_1} + \eta\mathrm{e}^L \le \frac{1}{16} + \frac{1}{16} = \frac{1}{8}$, so that $e_K^{(C)} \le \frac{3}{8}$.

It remains to read off the decision, for which a single coordinate no longer suffices. If $w \in \mathcal{L}$, every coordinate of x_K is at least 1, so a coordinate j with $(y_K^{(C)})_j \le 0$ satisfies $|(y_K^{(C)})_j - (x_K)_j| \ge 1$. Such coordinates form at most a fraction ξ of the n coordinates, where $\xi \le (e_K^{(C)})^p$ if $p < \infty$, since their contribution alone gives $n^{-1}\|y_K^{(C)} - x_K\|_p^p \ge \xi$, and $\xi = 0$ if $p = \infty$. In either case $\xi \le \frac{3}{8}$, because $e_K^{(C)} \le \frac{3}{8} < 1$ and $p \ge 1$.

At least $\frac{5}{8}$ of the coordinates of $y_K^{(C)}$ are therefore positive, and symmetrically at least $\frac{5}{8}$ are negative if $w \notin \mathcal{L}$. The table bound computes the sign of each coordinate, and approximate counting in AC^0 (Ajtai, 1990), applied to these signs, estimates the fraction of positive coordinates to within $\frac{1}{16}$. The estimate is thus at least $\frac{5}{8} - \frac{1}{16} > \frac{1}{2}$ if $w \in \mathcal{L}$ and at most $\frac{3}{8} + \frac{1}{16} < \frac{1}{2}$ if $w \notin \mathcal{L}$, so comparing it with $\frac{1}{2}$ decides $\mathcal{L}$. This count replaces the single-coordinate comparison at the end of Appendix A.2.4 and lies in $\mathsf{AC}^0 \subseteq \mathsf{D}$ for all three choices of D. Hence $\mathcal{L} \in \mathsf{D}$. □

Proposition 2 (Lower bound under ℓ_p)**.** *For every fixed $p \in [1, \infty]$, Theorem 2 remains true with the Lipschitz requirement in condition (b) replaced by L'-Lipschitz continuity in ℓ_p, for some constant L' depending only on $\mathcal{L}$. In the* $\mathsf{P/poly}$ *case its precision can be chosen to further satisfy* (1)*.*

Proof. We reuse the model u of Appendix A.3.2 as it stands: each coordinate of its condition is read by constantly many of the windows (I_j, J_j), and this bounded overlap already makes u Lipschitz continuous in every ℓ_p norm. All other conditions of Theorem 2, and acceptance with margin 1, are unaffected, since the model and the run are unchanged.

To compute the constant, write $\Delta\mathrm{ctx} := \mathrm{ctx} - \mathrm{ctx}'$ for the difference of two equal-length conditions and split it into its input part Δw and its endpoint-prediction part Δz. Since u ignores x and each $\bar{f}_j$ is 8-Lipschitz in the ℓ_∞ norm of its window, each output coordinate changes by at most 8 times the largest change inside its window. Raising to the p-th power, bounding that maximum by a sum, and summing over j give, for $1 \le p < \infty$,

$$\|u(x,t;\mathrm{ctx}) - u(x,t;\mathrm{ctx}')\|_p^p \le 8^p \sum_{j=1}^{n}\Big(\sum_{k\in I_j}|\Delta w_k|^p + \sum_{k\in J_j}|\Delta z_k|^p\Big).$$

In this double sum, each input coordinate appears in I_j for the κ coordinates j of its block, and each endpoint-prediction coordinate appears in J_j for at most 3κ coordinates j, those of its own block and of the two neighboring blocks. The double sum is therefore at most $\kappa\|\Delta w\|_p^p + 3\kappa\|\Delta z\|_p^p \le$

$3\kappa\|\Delta\mathrm{ctx}\|_p^p$, where the last step uses $\|\Delta\mathrm{ctx}\|_p^p = \|\Delta w\|_p^p + \|\Delta z\|_p^p$, so u is $8(3\kappa)^{1/p}$-Lipschitz in ℓ_p. The same argument with the maximum kept in place of the sum handles $p = \infty$, recovering the constant 8 of Theorem 2. As $8(3\kappa)^{1/p} \le 24\kappa$ for every $p \in [1, \infty]$, we may take $L' := 24\kappa$, which depends only on the alphabet of Lemma 4 and hence only on $\mathcal{L}$.

In the P/poly case, the parameter choice of Appendix A.3.2 sets $b(m) := \max\{3, \lceil 2\log_2 K(m)\rceil\}$, which does not depend on the Lipschitz constant and therefore still satisfies (1). □

B.3 Runge–Kutta Samplers

The upper bounds above are stated for vanilla flow matching with the Euler method. We extend them to any fixed explicit Runge–Kutta method. Let R be such a method with $q \in \mathbb{N}^+$ stages per step, whose tableau is specified by nodes $\tilde{\omega}_\ell \in [0, 1]$, weights $\tilde{\lambda}_\ell \in \mathbb{R}$, and coefficients $\tilde{a}_{\ell,k} \in \mathbb{R}$ for $1 \le k < \ell \le q$ (Hairer et al., 1993). Algorithm 4 gives the resulting sampler, RKFM, using the rounded tableau entries $\omega_\ell := \mathrm{round}_b(\tilde{\omega}_\ell)$, $\lambda_\ell := \mathrm{round}_b(\tilde{\lambda}_\ell)$, and $a_{\ell,k} := \mathrm{round}_b(\tilde{a}_{\ell,k})$. The Euler method is the special case $q = 1$, while Heun's method, another common flow-matching sampler (Karras et al., 2022; Lee et al., 2024), has $q = 2$.

As in Algorithm 3, we simulate RKFM by a parallel Picard iteration for R. The analysis of Appendices A.2.2 and A.2.4 then carries over with L scaled by a constant depending only on the tableau, at the cost of qK parallel model calls per round.

Algorithm 4 Runge–Kutta Flow Matching

Fixed: an explicit Runge–Kutta method R with constant stage count $q \in \mathbb{N}^+$ and rounded tableau entries $\omega_\ell \in F_b$, $\lambda_\ell \in F_b$, and $a_{\ell,k} \in F_b$

function RKFM($w; u, K$)
  $t_i \leftarrow \mathrm{round}_b(i/K)$ for $i = 0, \dots, K$
  $t_{i,\ell} \leftarrow \mathrm{round}_b(t_i + \omega_\ell h_i)$ for $i = 0, \dots, K-1$; $\ell = 1, \dots, q$
  Sample $\tilde{x}_0 \sim p_{\text{prior}}$; $x_0 \leftarrow \mathrm{round}_b(\tilde{x}_0)$
  for $i = 0, \dots, K-1$ **do**
    for $\ell = 1, \dots, q$ **do**
      $Y_{i,\ell} \leftarrow \mathrm{round}_b\big(x_i + h_i \sum_{k<\ell} a_{\ell,k} v_{i,k}\big)$
      $v_{i,\ell} \leftarrow u(Y_{i,\ell}, t_{i,\ell}; w)$
    end for
    $x_{i+1} \leftarrow \mathrm{round}_b\big(x_i + h_i \sum_{\ell=1}^{q} \lambda_\ell v_{i,\ell}\big)$
  end for
  return x_K
end function

Proposition 3 (Runge–Kutta samplers)**.** *Let q be a constant integer and* R *be any fixed explicit Runge–Kutta method with q stages, whose nodes lie in $[0, 1]$. There is a constant $\gamma_{\mathsf{R}} \ge 1$, depending only on* R*, such that the following hold with* VanillaFM *replaced by* RKFM*:*

- *Lemma 1 remains true once the precision condition* (2) *is replaced by*

$$b(m) \ge \log_2 K(m) + \gamma_{\mathsf{R}}(\log_2 \mathrm{e}) L(m) + c_0 \qquad \text{for every } m, \tag{13}$$

  where the constant $c_0 = c_0(B, \mathsf{R})$ now also depends on R*, and the bound $K(m)$ on the parallel calls per round is replaced by $qK(m)$.*
- *Corollary 1 remains true once the lower bound in* (12) *is replaced by* (13)*.*
- *Theorem 1 remains true.*

Proof. We simulate the Runge–Kutta sampler of Algorithm 4 by the parallel Picard iteration of Algorithm 5, following the same strategy as the Euler case: write the RKFM trajectory as a fixed-point system, solve it by Algorithm 5, and bound the error with Lemma 2. The only new feature is that each step has q stages, so the system has qK positions instead of K. Each stage depends on earlier stages in the same step as well as on all stages of previous steps.

The fixed-point system. Fix w and couple the two algorithms through a common initial x_0. Let $Y^*_{i,\ell}$ and x^*_i denote the true stages and states of Algorithm 4, and write $v^*_{i,\ell} := u(Y^*_{i,\ell}, t_{i,\ell}; w)$. Since

Algorithm 5 Parallel Runge–Kutta Picard Inference

Fixed: an explicit Runge–Kutta method R with constant stage count $q \in \mathbb{N}^+$ and rounded tableau entries $\omega_\ell \in F_b$, $\lambda_\ell \in F_b$, and $a_{\ell,k} \in F_b$

function PARALLELRK($w; u, K, C$)
  $t_i \leftarrow \mathrm{round}_b(i/K)$ for $i = 0, \dots, K$
  $t_{i,\ell} \leftarrow \mathrm{round}_b(t_i + \omega_\ell h_i)$ for $i = 0, \dots, K-1$, $\ell = 1, \dots, q$
  Sample $\tilde{x}_0 \sim p_{\text{prior}}$; $x_0 \leftarrow \mathrm{round}_b(\tilde{x}_0)$
  $Y^{(0)}_{i,\ell} \leftarrow x_0$ for all i, ℓ; $X^{(0)} \leftarrow x_0$
  for $r = 0, \dots, C-1$ **do**
    in parallel for all i, ℓ: $v^{(r)}_{i,\ell} \leftarrow u(Y^{(r)}_{i,\ell}, t_{i,\ell}; w)$
    $S^{(r)}_i \leftarrow \sum_{j=0}^{i-1} \sum_{k=1}^{q} \mathrm{round}_b(\lambda_k v^{(r)}_{j,k})$ for $i = 0, \dots, K$
    $T^{(r)}_{i,\ell} \leftarrow \sum_{k=1}^{\ell-1} \mathrm{round}_b(a_{\ell,k} v^{(r)}_{i,k})$ for all i, ℓ
    in parallel for all i, ℓ: $Y^{(r+1)}_{i,\ell} \leftarrow \mathrm{round}_b\big(x_0 + \big(S^{(r)}_i + T^{(r)}_{i,\ell}\big)/K\big)$
    $X^{(r+1)} \leftarrow \mathrm{round}_b\big(x_0 + S^{(r)}_K/K\big)$
  end for
  return $X^{(C)}$
end function

$\omega_\ell \in [0,1]$ and round_b is nondecreasing and fixes F_b, every stage time satisfies $t_{i,\ell} \in [t_i, t_{i+1}] \subseteq [0,1]$, so the boundedness and Lipschitz hypotheses apply at every stage. Each round_b displayed in Algorithms 4 and 5 evaluates its argument over the reals and rounds the result once. Unrolling the rounded updates gives the fixed-point equations

$$Y^*_{i,\ell} = x_0 + \sum_{j<i} h_j \sum_{k=1}^{q} \lambda_k v^*_{j,k} + h_i \sum_{k<\ell} a_{\ell,k} v^*_{i,k} + \rho_{i,\ell}, \qquad \|\rho_{i,\ell}\|_\infty \le \frac{K\delta}{2}, \tag{14}$$

and

$$x^*_K = x_0 + \sum_{j=0}^{K-1} h_j \sum_{k=1}^{q} \lambda_k v^*_{j,k} + \rho_K, \qquad \|\rho_K\|_\infty \le \frac{K\delta}{2}. \tag{15}$$

We now index all positions in dependency order. Put index 0 for x_0, index $\nu = qi + \ell$ for stage $Y^*_{i,\ell}$ with $0 \le i \le K-1$ and $1 \le \ell \le q$, and index $M := qK + 1$ for the endpoint x^*_K. For each $\nu \ge 1$, write y^*_ν for the corresponding quantity (the stage $Y^*_{i,\ell}$ when $\nu = qi + \ell$, and the endpoint x^*_K when $\nu = M$), and for $1 \le \nu \le qK$ write v^*_ν for the corresponding stage velocity $v^*_{i,\ell}$. Then the fixed-point equations for all positions can be written uniformly as

$$y^*_\nu = x_0 + \sum_{1 \le \nu' < \nu} \Lambda_{\nu,\nu'}\, v^*_{\nu'} + \rho_\nu, \qquad \|\rho_\nu\|_\infty \le \frac{K\delta}{2},$$

where, by (14) and (15), each coefficient $\Lambda_{\nu,\nu'}$ is either 0, some $h_j\lambda_k$, or some $h_i a_{\ell,k}$. Put $\mu := 3\max_{\ell,k}\{1, |\tilde{\lambda}_\ell|, |\tilde{a}_{\ell,k}|\}$, a constant depending only on the tableau, so that every rounded tableau entry has magnitude at most $\mu/2$. The precision condition (13) gives $\delta \le 1/K$, so $h_j \le 2/K$, and hence every coefficient satisfies

$$|\Lambda_{\nu,\nu'}| \le \frac{\mu}{K}.$$

Error analysis. For each position ν, define $e^{(r)}_\nu$ as follows: if $\nu = qi + \ell$ is a stage, then $e^{(r)}_\nu := \|Y^{(r)}_{i,\ell} - Y^*_{i,\ell}\|_\infty$; if $\nu = M$ is the endpoint, then $e^{(r)}_M := \|X^{(r)} - x^*_K\|_\infty$; and $e^{(r)}_0 := 0$, since both runs share x_0. Put $s_\nu := \nu/M$, $\gamma_{\mathsf{R}} := 2q\mu$, $B_{\mathsf{R}} := 2q\mu B$, and $A_{\mathsf{R}} := q(\mu B + 1) + 1$, constants depending only on B and R. Since the iteration starts every position at x_0, bounding the at most ν velocities in the uniform equation by B gives

$$e^{(0)}_\nu = \|x_0 - y^*_\nu\|_\infty \le \sum_{1 \le \nu' < \nu} |\Lambda_{\nu,\nu'}|\, \|v^*_{\nu'}\|_\infty + \|\rho_\nu\|_\infty \le \nu \cdot \frac{\mu B}{K} + \frac{K\delta}{2} \le B_{\mathsf{R}} s_\nu + A_{\mathsf{R}} K\delta,$$

the last step because $\nu/K = s_\nu M/K \le 2qs_\nu$.

For the inductive step, write $y_\nu^{(r)}$ and $v_\nu^{(r)}$ for the iterate and velocity of Algorithm 5 at position ν, in the indexing above, and let $\Lambda^\circ_{\nu,\nu'}$ be the nominal coefficients, obtained from $\Lambda_{\nu,\nu'}$ by replacing the step h_j or h_i with $1/K$. Each rounded tableau entry has magnitude at most $\mu/2$, so $|\Lambda^\circ_{\nu,\nu'}| \le \mu/(2K)$, and $|1/K - h_j| \le \delta$ bounds the mismatch by $|\Lambda^\circ_{\nu,\nu'} - \Lambda_{\nu,\nu'}| \le \mu\delta/2$. Expanding the update of Algorithm 5 in the same indexing,

$$y_\nu^{(r+1)} = x_0 + \sum_{1\le\nu'<\nu} \Lambda^\circ_{\nu,\nu'} v_{\nu'}^{(r)} + \rho_\nu^{(r)}, \qquad \|\rho_\nu^{(r)}\|_\infty \le \frac{(2q+1)\delta}{2},$$

where $\rho_\nu^{(r)}$ collects the roundings of the at most qK products in $S_i^{(r)}$ and the at most q products in $T_{i,\ell}^{(r)}$, each of size at most $\delta/2$ and divided by K, together with the outer rounding. Subtracting the uniform equation from this expansion and taking norms,

$$e_\nu^{(r+1)} \le \sum_{1\le\nu'<\nu} |\Lambda^\circ_{\nu,\nu'}|\, \|v_{\nu'}^{(r)} - v_{\nu'}^*\|_\infty + \sum_{1\le\nu'<\nu} |\Lambda^\circ_{\nu,\nu'} - \Lambda_{\nu,\nu'}|\, \|v_{\nu'}^*\|_\infty + \|\rho_\nu^{(r)}\|_\infty + \|\rho_\nu\|_\infty.$$

In the first sum, the two velocities at position ν' share the stage time and the condition, so Lipschitz continuity gives $\|v_{\nu'}^{(r)} - v_{\nu'}^*\|_\infty \le L e_{\nu'}^{(r)}$, and $|\Lambda^\circ_{\nu,\nu'}| \le \mu/(2K) \le \gamma_{\mathsf{R}}/M$ because $M \le 2qK$. The second sum has fewer than $M \le 2qK$ terms, each at most $\frac{\mu\delta}{2}B$, so it is at most $q\mu BK\delta$. The two residuals add at most $(2q+1)\delta/2 + K\delta/2 \le (q+1)K\delta$. Hence, by the choice of A_{R},

$$e_\nu^{(r+1)} \le \frac{\gamma_{\mathsf{R}} L}{M} \sum_{\nu'<\nu} e_{\nu'}^{(r)} + A_{\mathsf{R}} K\delta.$$

This is exactly the recurrence of Lemma 2, with its parameters set to M, B_{R}, $\gamma_{\mathsf{R}}L$, $\alpha = A_{\mathsf{R}}K\delta$, and $\beta = A_{\mathsf{R}}K\delta$. Applying that lemma with $C := \lceil \mathrm{e}\gamma_{\mathsf{R}}L + 4B_{\mathsf{R}} \rceil$ gives

$$e_M^{(C)} \le \frac{1}{4} + A_{\mathsf{R}}K\delta\big(1 + \mathrm{e}^{\gamma_{\mathsf{R}}L}\big).$$

The precision condition (13) implies $K\delta \le 2^{-c_0}\mathrm{e}^{-\gamma_{\mathsf{R}}L}$. Choosing $c_0 = c_0(B, \mathsf{R})$ with $2A_{\mathsf{R}}2^{-c_0} \le \frac{1}{4}$ makes $e_M^{(C)} \le \frac{1}{2}$. Thus u accepts $\mathcal{L}$ with margin $\frac{1}{2}$ under Algorithm 5 whenever it accepts $\mathcal{L}$ with margin 1 under RKFM, and the algorithm runs for $C = O(L)$ rounds with qK parallel model calls per round. This proves the first item.

Expressiveness upper bounds. For the analogue of Theorem 1, where $b = O(\log m)$, we split as in Appendix A.2.4, now at $b \ge \log_2 K + \gamma_{\mathsf{R}}(\log_2 \mathrm{e})L + c_1$. The small-$K$ case again re-runs the sampler directly, now making q stage calls per step, and lies in D by the same argument as in Appendix A.2.4. In the large-K case, we replace the normalized prefix sums by the approximate ones of Lemma 3, applied with $N := qK$ and accuracy $\varepsilon := \eta/q$ to the rounded products $\mathrm{round}_b(\lambda_k v_{j,k}^{(r)})$, which are bounded by a constant, and rescaled by the integer q, as in Appendix A.2.4. All remaining updates depend on only $O(1)$ scalars of $O(\log m)$ bits. With the time grid and stage times hardwired, they can be computed exactly by the table bound. The approximation adds η to the additive constant of the recurrence, so Lemma 2, now with $\beta = A_{\mathsf{R}}K\delta + \eta$, gives

$$e_M^{(C)} \le \frac{1}{4} + A_{\mathsf{R}}K\delta + \big(A_{\mathsf{R}}K\delta + \eta\big)\mathrm{e}^{\gamma_{\mathsf{R}}L}.$$

Taking $\eta := \frac{1}{8}\mathrm{e}^{-\gamma_{\mathsf{R}}L}$ and choosing $c_1 = c_1(B, \mathsf{R})$ with $2A_{\mathsf{R}}2^{-c_1} \le \frac{1}{8}$ gives $e_M^{(C)} \le \frac{1}{2}$. Since $A_{\mathsf{R}} \ge q$, this choice also gives $\delta \le K\delta \le 2^{-c_1}\mathrm{e}^{-\gamma_{\mathsf{R}}L} \le \eta/q$, as Lemma 3 requires. The size and depth argument of Appendix A.2.4 then places $\mathcal{L}$ in D, which proves the third item.

For the analogue of Corollary 1, where $b = \mathrm{poly}(m)$, arithmetic with the hardwired rounded tableau entries and the exact prefix sums lies in TC^0 (Vollmer, 1999; Hesse et al., 2002), so the depth is $O(L)$ as in Corollary 1. This proves the second item and hence the proposition. □

B.4 MATCHING THE CIRCUIT DEPTH

Corollary 1 shows that any run of VANILLAFM can be compressed into $O(L)$ sequential rounds. Here we show that these rounds match circuit depth up to constant factors: given any polynomial-size target majority circuit of depth d, VANILLAFM can simulate it using a model with Lipschitz constant $O(d)$, while each model evaluation can be computed by a constant-depth circuit. The construction proceeds in three phases, determined by the current time t: resetting the sampler state, evaluating one layer of the target circuit per sampling step, and finally amplifying the resulting verdict to satisfy the acceptance margin.

Proposition 4 (Encoding majority circuits into VANILLAFM)**.** *Let $\{d(m)\}_{m=1}^{\infty}$ be a sequence of positive integers, and $\mathcal{L}$ be a language decided by polynomial-size majority circuits of depth $d(m)$. Then there are $K(m) = \Theta(d(m))$, $n(m) = \mathrm{poly}(m)$, and $b(m) = \mathrm{poly}(m)$ satisfying* (2) *with $B = 8$ and $L(m) := 8K(m)$, and a model u, computed by TC^0 circuits, such that u is 8-bounded and $8K(m)$-Lipschitz continuous and accepts $\mathcal{L}$ with margin 1 under* VANILLAFM.

Proof. Every gate of the circuit is a threshold gate with weights in $\{-1, 1\}$ and an integer threshold. An AND, OR, or MAJORITY gate of fan-in r has unit weights and threshold r, 1, or $\lceil r/2 \rceil$, respectively, and a NOT gate has weight -1 and threshold 0.

We construct the model u explicitly. Recode Boolean values by $0 \mapsto -1$ and $1 \mapsto 1$. A gate keeps its ± 1 weights $q_1, \ldots, q_r$, and its threshold θ_{old} becomes the integer $\theta := 2\theta_{\text{old}} - \sum_{j=1}^{r} q_j$. Index the state coordinates by the gates, let n be the number of gates, and write $*$ for the output gate. The run stores the output of each gate γ in the state coordinate x_γ. The layer of a gate γ is its depth $\ell(\gamma) \in \{1, \ldots, d\}$. Every gate reads only primary inputs and gates of earlier layers.

Let K be the least power of two with $K \geq 8(d+1)$, so that $d + 1 \leq K/8$ and $K \leq 16(d+1)$. We will choose $b \geq \log_2 K$ at the end of the proof. Since K is a power of two and round_b fixes F_b, the time grid is exact: $t_i = i/K$ and $h_i = 1/K$.

Encodings and gate extensions. Write $\mathrm{clip}(y; c) := \max\{-c, \min\{c, y\}\}$ for the truncation of a scalar y to $[-c, c]$. Gate outputs are stored at scale $1/K$: one Euler step with velocity ± 1 writes exactly $\pm 1/K$ into a coordinate. A state coordinate x_γ is therefore decoded by $\mathrm{clip}(Kx_\gamma; 1)$, which maps the stored encodings $\pm 1/K$ to ± 1. An input bit $w_j \in \{0, 1\}$ is decoded by $\mathrm{clip}(2w_j - 1; 1)$, which maps 0 to -1 and 1 to 1.

Consider a gate with weights $q_1, \ldots, q_r \in \{-1, 1\}$ and integer threshold θ, and set $k := \lceil (\theta + r)/2 \rceil$. On a vertex $a \in \{-1, 1\}^r$, the gate fires if and only if $\sum_j q_j a_j = 2\,|\{j : q_j a_j = 1\}| - r$ reaches θ, that is, if and only if at least k of the ± 1 values $q_j a_j$ equal 1. The recoded gate value is therefore the k-th largest of the $q_j a_j$ when $1 \leq k \leq r$, and a constant otherwise. Accordingly, extend the gate to $[-1, 1]^r$ by letting $\hat{g}(y)$ be the k-th largest of $q_1 y_1, \ldots, q_r y_r$, with $\hat{g} \equiv 1$ if $k \leq 0$ and $\hat{g} \equiv -1$ if $k > r$. Then $\hat{g}$ equals the recoded gate value on the vertices, and every value of $\hat{g}$ is one of the $q_j y_j$ or ± 1, so $\hat{g}$ maps F_b^r into F_b and the model needs no rounding.

The model. Let $G_\gamma(w, x)$ denote $\hat{g}$ of gate γ applied to its decoded inputs. We define u coordinatewise by

$$\big(u(x, t; w)\big)_\gamma := \begin{cases} -\,\mathrm{clip}(Kx_\gamma; 8) & t = i/K \text{ with } 0 \leq i < K/4, \\ G_\gamma(w, x) & t = (K/4 + \ell(\gamma) - 1)/K, \\ 8\,\mathrm{clip}(Kx_*; 1) & t = i/K \text{ with } K/4 + d \leq i < K, \\ 0 & \text{otherwise}, \end{cases} \tag{16}$$

on valid inputs with parameters $(n(m), b(m), m)$, and $u(x, t; w) := \mathbf{0}^n$ on all other encoded inputs.

The run. We now show that u accepts $\mathcal{L}$ with margin 1 under VANILLAFM. In the first phase, one step maps x_γ to 0 if $|x_\gamma| \leq 8/K$ and otherwise moves it toward 0 by $8/K$, so from $\|x_0\|_\infty \leq 1$ the state reaches $\mathbf{0}$ within $K/8 + 1 \leq K/4$ steps and stays there.

The second phase evaluates one circuit layer per step. A coordinate has zero velocity at every second-phase step except its own layer step, so a value once written persists. By construction, step $K/4 + \ell - 1$ writes the value of every layer-ℓ gate, encoded as $\pm 1/K$, into its coordinate.

The third phase amplifies the verdict: after step $K/4 + d - 1$ every gate value is written, so the decoded output $\mathrm{clip}(Kx_*; 1)$ is the verdict ± 1. The shared velocity $8\,\mathrm{clip}(Kx_*; 1)$ stays pinned at ± 8 for the $K - K/4 - d \geq \frac{5}{8}K$ remaining steps. Every final coordinate is then at least $8 \cdot \frac{5}{8} - \frac{1}{K} \geq 1$ on acceptance and at most -1 on rejection, so u accepts $\mathcal{L}$ with margin 1.

The run is exact: every velocity lies in $\{0, \pm 1, \pm 8\}$ or equals $-Kx_\gamma$, so every Euler increment is 0, $\pm 1/K$, $\pm 8/K$, or $-x_\gamma$, all in F_b.

Lipschitz constant and circuit complexity. We first check that $\hat{g}$ is 1-Lipschitz. Suppose that $\|y - y'\|_\infty \leq \varepsilon$, so that $q_j y'_j \leq q_j y_j + \varepsilon$ for every j. The k-th largest is nondecreasing in every coordinate and grows by exactly ε when every coordinate grows by ε. Hence $\hat{g}(y') \leq \hat{g}(y) + \varepsilon$, and exchanging y and y' gives the reverse bound.

The model is 8-bounded. For fixed t, the velocity is K-Lipschitz in the reset phase and $8K$-Lipschitz in the amplification phase, as clip is 1-Lipschitz. In the layer phase, the velocity is K-Lipschitz: the state decoder $\mathrm{clip}(Kx_\gamma; 1)$ is K-Lipschitz, the input decoder $\mathrm{clip}(2w_j - 1; 1)$ is 2-Lipschitz, and $\hat{g}$ is 1-Lipschitz. Since the Lipschitz condition of §2.2 constrains each t separately, u is $8K$-Lipschitz continuous.

It remains to check that u is computed by TC^0 circuits. Every scalar involved occupies polynomially many bits in the input length. Comparisons, additions, and multiplexing lie in AC^0 (Vollmer, 1999), and multiplication by K is a bit shift, as K is a power of two.

The only nontrivial computation is $\hat{g}$, the k-th largest of the r scalars $q_j y_j$. This value is the largest candidate $c \in \{q_1 y_1, \ldots, q_r y_r\}$ with $|\{j : q_j y_j \geq c\}| \geq k$. The circuit compares all pairs of the scalars in parallel, tests each candidate by one threshold gate over its r comparison bits, and selects the largest passing candidate by multiplexing. These blocks have constant depth and polynomial size, so u is computed by TC^0 circuits.

Parameters. Finally, we fix the parameters. The quantities $n(m)$ and $K(m) \leq 16(d(m) + 1)$ are $\mathrm{poly}(m)$, and

$$b(m) := \left\lceil \log_2 K(m) + 8(\log_2 \mathrm{e})K(m) + c_0(8) \right\rceil = \mathrm{poly}(m)$$

satisfies $b \geq \log_2 K$ and (2) with $B = 8$ and $L(m) = 8K(m)$. □

Remark. Proposition 4 and Corollary 1 show that the $O(L)$ rounds of Lemma 1 cannot be improved in general without collapsing the depth hierarchy of majority circuits. Suppose that some inference algorithm simulates VANILLAFM in $f(L) = o(L)$ rounds, each consisting of polynomially many parallel calls to u and TC^0 arithmetic. The model of Proposition 4 has $L(m) = \Theta(d(m))$ and is computed by majority circuits of constant depth, so polynomial-size majority circuits of depth $d(m)$ would collapse to polynomial-size majority circuits of depth $O(f(L(m))) = o(d(m))$, which is unlikely. For example, the depth hierarchy of majority circuits within TC^0 would collapse to a constant depth, whereas it is widely conjectured to be infinite (Hajnal et al., 1993).

C Formalization and Circuit Complexity of the Tasks in §4.2

This appendix formalizes the three synthetic tasks introduced in §4.2 and establishes the circuit-complexity claims used there. Specifically, we show that line tracing is NC^1-complete, while long multiplication and long text rendering are TC^0-complete, matching the separation barriers between VanillaFM and SelfCondFM in §3.1 for TC^0 and AC^0 backbones, respectively. As in Appendix A.1, all circuit classes considered below are non-uniform.

C.1 Task Descriptions

We first describe the three task families before giving their formal definitions and complexity analyses. The tasks are illustrated in Figure 1.

Line tracing. The input is an image containing five black polylines that traverse m permutation stages from left to right, with one polyline starting in each of the five rows. At stage k, a permutation $g_k \in A_5$ reroutes the polyline entering row i to row $g_k(i)$, where A_5 denotes the group of even permutations on five elements. The target is a colored version of the input in which each polyline is assigned a fixed color determined by its starting row. Whenever two polylines cross within a stage, we resolve occlusion by rendering the polyline entering the stage from the higher row on top.

Long multiplication. The input is an image containing a multiplication problem of two positive integers A and B, each with at most m digits and no leading zeros. The two integers are rendered with their digits aligned by column and their least significant digits in the rightmost column. The target is a complete step-by-step long-multiplication worksheet: writing $B = \sum_i B_i 10^i$, it displays the shifted partial products AB_i10^i, the digitwise carries, their vertical summation, and the final product AB. The worksheet follows a digit-aligned grid layout, with each digit rendered in a separate cell and carry digits placed at the upper left of the corresponding cells. Additional line segments indicate the summation of the partial products.

Long text rendering. The input is a text passage of m words sampled from the TinyStories (Eldan & Li, 2023) dataset, given to the model as a text condition. The target is a monospaced rendering of the passage on a canvas of fixed usable width W. The first word begins at the left margin; each subsequent word is placed on the current line if it fits after a single interword space, and otherwise starts a new line. At each paragraph break, one line is left blank, and the next paragraph begins at the left margin.

C.2 Preliminaries for the Complexity Analysis

Next, we formalize these three image-space tasks and establish their circuit complexities through their underlying symbolic functions.

Completeness for functions. Each of the three tasks produces a long output rather than a single bit, so we phrase completeness through the output bits. For a function f whose output has polynomial length, put

$$\mathrm{Bit}(f) := \{(x, j) : \text{the } j\text{-th bit of } f(x) \text{ is } 1\}.$$

As in Theorem 1, we write D for a circuit class. We call f D-*complete* when $\mathrm{Bit}(f) \in \mathsf{D}$ and every language in D is decided by AC^0 circuits with oracle gates for $\mathrm{Bit}(f)$, that is, when f is complete for D under AC^0 Turing reductions. The three propositions below use this notion with $\mathsf{D} = \mathsf{NC}^1$ for line tracing and $\mathsf{D} = \mathsf{TC}^0$ for the other two tasks.

Symbolic and image-space tasks. Line tracing and long multiplication map images to images, whereas long text rendering maps a word sequence with paragraph boundaries to an image. We call these three image-output mappings *image-space tasks*, distinguishing them from their underlying symbolic functions. For each task, we establish the circuit complexity of its symbolic function, then give bidirectional AC^0 Turing reductions between that function and the corresponding image-space task.

Asymptotic interpretation. The propositions below concern asymptotic task families as m grows. In practice, our experiments in §4.2 instantiate these families over a finite range of problem sizes m and report a single score for each task. A line-tracing instance with $m = 50$ contains every shorter prefix length as a subproblem. Long multiplication draws operands of five to seven digits, and long text rendering draws passages of varying word count m. Circuit complexity is a property of the family, so the propositions describe the computation that the tasks demand as m grows.

C.3 Line Tracing

A line-tracing instance, as illustrated in the left column of Figure 1, can be viewed as a sequence $(g_1, \ldots, g_m) \in A_5^m$, where A_5 denotes the alternating group on five elements. Here g_k reroutes the polyline entering row i of stage k to row $g_k(i)$. With e the identity of A_5, set

$$\pi_0 = e, \qquad \pi_k = g_k \circ \pi_{k-1} \quad (1 \le k \le m),$$

so that the polyline starting in row i occupies row $\pi_k(i)$ after k stages. Each polyline keeps the color c_i of its starting row i, so row j carries the color $c_{\pi_k^{-1}(j)}$ after k stages. The colored target is exactly the sequence of prefix permutations $\mathrm{Trace}_{A_5}(g_1, \ldots, g_m) := (\pi_0, \ldots, \pi_m)$.

Proposition 5 (Line tracing)**.** *The line-tracing task, mapping a black-and-white input image to its colored target, is* NC^1*-complete under* AC^0 *Turing reductions.*

Proof. We first consider the symbolic function Trace_{A_5}. Since A_5 is fixed, multiplying two encoded permutations is a constant-size lookup, so a balanced tree computes any π_k with depth $O(\log m)$. Computing all m prefix products independently still uses only polynomial size, so the output-bit relation of Trace_{A_5} is in NC^1.

To establish NC^1-hardness, we use Barrington's result that the word problem for every fixed non-solvable group is NC^1-complete under AC^0 reductions (Barrington, 1986). Since A_5 is non-solvable, that result applies to the word problem of deciding whether $g_m \circ \cdots \circ g_1 = e$. An encoded group element occupies a constant number of bits, so a constant number of oracle queries returns the final component π_m of the trace, and an AC^0 test compares it with e. Hence Trace_{A_5} is NC^1-complete.

Parallel local lookups decode $(g_k)_{k=1}^m$ from the black-and-white stage templates and $(\pi_k)_{k=0}^m$ from the colored target, and conversely render both sequences as images. These conversions lie in AC^0, so the classification transfers to the line-tracing task. □

C.4 Long Multiplication

We write $\mathrm{LMul}(A, B)$ for the complete decimal worksheet of two positive integers A and B, shown in the middle column of Figure 1. For $B = \sum_i B_i 10^i$, put $P_i = AB_i 10^i$ and $S_i = \sum_{j=0}^{i} P_j$. The worksheet contains the shifted partial products P_i, the successive partial sums S_i for $i \ge 1$, the digitwise carries used in these multiplications and additions, and the final product AB, all in decimal.

Proposition 6 (Long multiplication)**.** *The long-multiplication task, mapping an operand image to its rendered worksheet, is* TC^0*-complete under* AC^0 *Turing reductions.*

Proof. We first analyze the symbolic worksheet function LMul. To show that $\mathrm{LMul} \in \mathsf{TC}^0$, recall that fixed-base conversion, iterated addition, multiplication, division, and remainder are computable by constant-depth threshold circuits (Chandra et al., 1984; Hesse et al., 2002). These operations compute every P_i, every S_i, and every decimal output digit in parallel, and each displayed carry is obtained from the corresponding fixed-base quotient and remainder. The number of worksheet entries and the length of each entry are polynomial in the input length, so the complete worksheet is in TC^0.

We now prove TC^0-hardness. For $x = (x_0, \ldots, x_{N-1}) \in \{0,1\}^N$, the exact-counting function is

$$\mathrm{Count}_N(x) := \sum_{i=0}^{N-1} x_i,$$

with its output written in binary. This function is TC^0-complete under AC^0 Turing reductions (Chandra et al., 1984). To establish TC^0-hardness, it therefore suffices to reduce Count_N to LMul.

Given x, we set

$$d = \lceil \log_{10}(N+2) \rceil, \qquad Q = 10^d,$$

and define the positive decimal integers A and B by

$$A = 1 + \sum_{i=0}^{N-1} x_i Q^{i+1}, \qquad B = \sum_{j=0}^{N-1} Q^j.$$

Their decimal encodings consist of d-digit blocks and are produced by an AC^0 circuit, since every output digit is either a constant or an input bit x_i. When AB is written in base Q, every coefficient is at most $N + 1 < Q$, so no carry crosses a d-digit block, and the coefficient of Q^N is exactly $\mathrm{Count}_N(x)$.

The final row of the worksheet contains AB, from which an AC^0 readout extracts this block. Its $d = O(\log N)$ decimal digits have an $O(\log N)$-bit encoding. A non-uniform lookup on these bits has $2^{O(\log N)} = N^{O(1)}$ entries and converts the block to binary in AC^0. Thus Count_N, and hence every language in TC^0, reduces to LMul.

The fixed digit glyphs and worksheet grid similarly allow parallel local lookups to decode (A, B) from the operand image and $\mathrm{LMul}(A, B)$ from the worksheet image, and conversely to render both as images. These conversions lie in AC^0, so the same classification holds for the long-multiplication task. □

C.5 Long Text Rendering

As illustrated in the right column of Figure 1, long text rendering uses a fixed monospaced raster font over a finite alphabet Σ with $|\Sigma| \geq 2$. We call the font nontrivial if its glyph-raster map is not constant, that is, if some two characters render differently. We normalize horizontal distances to character cells, so the width budget is an integer $W \geq 3$.

An input consists of a sequence $((s_i, p_i))_{i=1}^m$ where $s_i \in \Sigma^+$ is the i-th word and $p_i \in \{0, 1\}$ is a paragraph-boundary indicator. By convention, $p_1 = 1$. For $i > 1$, $p_i = 1$ means that s_i begins a new paragraph after one blank line, whereas $p_i = 0$ means that it continues the current paragraph.

The words s_i determine the glyph content, while the auxiliary wrapping function below determines their positions. Let $a_i := |s_i| \in \{1, \ldots, W\}$ be the length of word i. We define three auxiliary variables for each word: ℓ_i is its line index, starting from 1; ζ_i is its horizontal starting position, starting from 0; and r_i is the occupied width of the line after placing s_i, equivalently the exclusive right boundary of s_i.

With $(\ell_1, \zeta_1, r_1) := (1, 0, a_1)$, we define, for $i > 1$,

$$(\ell_i, \zeta_i, r_i) = \begin{cases} (\ell_{i-1} + 2,\ 0,\ a_i), & p_i = 1, \\ (\ell_{i-1},\ r_{i-1} + 1,\ r_{i-1} + 1 + a_i), & p_i = 0 \text{ and } r_{i-1} + 1 + a_i \leq W, \\ (\ell_{i-1} + 1,\ 0,\ a_i), & p_i = 0 \text{ and } r_{i-1} + 1 + a_i > W, \end{cases}$$

and denote the sequence $((\ell_i, \zeta_i))_{i=1}^m$ by $\mathrm{Wrap}_W\big((a_i, p_i)_{i=1}^m\big)$. The three cases are the greedy rule stated in the task descriptions above, where a word joins the current line whenever it fits with a single interword space.

The long text rendering task is the function Render_W. On input $((s_i, p_i))_{i=1}^m$, it returns the full-height rasterization of these words under the rule above. Its canvas has W character cells per line and $2m$ line slots, with unused slots left white. The largest line index is $2m - 1$, so the canvas accommodates every input and has a fixed output length given m.

Proposition 7 (Long text rendering)**.** *For every fixed $W \geq 3$ and fixed nontrivial monospaced raster font over an alphabet Σ with $|\Sigma| \geq 2$, the function Render_W is TC^0-complete under AC^0 Turing reductions.*

Proof. We first classify the auxiliary wrapping function. To show that $\mathrm{Wrap}_W \in \mathsf{TC}^0$, we track the occupied width with a finite automaton. For each input pair (a, p), its transition on the state set $\{1, \ldots, W\}$ is

$$\tau_{a,p}(r) = \begin{cases} a, & p = 1, \\ r + 1 + a, & p = 0 \text{ and } r + 1 + a \le W, \\ a, & p = 0 \text{ and } r + 1 + a > W, \end{cases}$$

and we let M_W be the transition monoid generated under composition by $\{\tau_{a,p} : 1 \le a \le W,\ p \in \{0, 1\}\}$.

For any initial state r_0, reading the first i input pairs yields

$$r_i = (\tau_{a_i,p_i} \circ \cdots \circ \tau_{a_1,p_1})(r_0).$$

We arrange the states in cyclic order. For $p = 0$, the list $\tau_{a,0}(1), \ldots, \tau_{a,0}(W)$ advances in that order, wraps around at most once, and may repeat a; for $p = 1$, the transition is constant. Both are orientation-preserving transformations of the cycle, and such transformations are closed under composition.

Every subgroup G of M_W is cyclic. To see this, let e be the identity of G and $S = e(\{1, \ldots, W\})$. Although e need not fix every state, it fixes every element of S. Since $g = g \circ e$, each $g \in G$ is determined by its restriction to S, where it acts as a cyclic-order-preserving permutation. Restriction to S therefore embeds G into the rotation group of S. Since this rotation group is cyclic, so is G. In particular, every subgroup of M_W is solvable.

By the characterization of regular languages recognized by finite monoids with solvable subgroups (Barrington et al., 1992), every state predicate of this automaton lies in $\mathsf{ACC}^0 \subseteq \mathsf{TC}^0$. Applying these state-predicate circuits to every prefix gives a one-hot encoding of every r_i, still with constant depth and polynomial size. The first word needs no special treatment: since $p_1 = 1$, the map $\tau_{a_1,1}$ is constant with value a_1, so applying it to any initial state returns $r_1 = a_1$.

For each $i \ge 2$, the automatic-wrap indicator χ_i and the total line increment Δ_i are

$$\chi_i := (1 - p_i)\mathbf{1}\{r_{i-1} + 1 + a_i > W\}, \qquad \Delta_i := \chi_i + 2p_i,$$

and computing them requires only a constant-size lookup once r_{i-1}, a_i, and p_i are known. Threshold circuits compute all prefix sums

$$\ell_i = 1 + \sum_{j=2}^{i} \Delta_j$$

in constant depth, and the starting position is $\zeta_i = 0$ when $\chi_i + p_i > 0$ and $\zeta_i = r_{i-1} + 1$ otherwise. Thus the entire wrapping function is in TC^0.

To establish TC^0-hardness, we use the single-paragraph subfamily, setting $p_1 = 1$ and $p_i = 0$ for every $i > 1$, and reduce MAJORITY to a single query of Wrap_W. Given $x \in \{0, 1\}^N$, we begin with a sentinel word of width W, so that the first line is full, and encode each input bit x_i by the two-word gadget $\gamma(x_i)$ given by

$$\gamma(0) := (1, W - 2), \qquad \gamma(1) := (W, W).$$

The gadgets $\gamma(x_0), \ldots, \gamma(x_{N-1})$ are appended after the sentinel.

Whenever a gadget begins, the current line is full. The 0-gadget starts one new line and fills it exactly, because $1 + 1 + (W - 2) = W$, while the 1-gadget starts two new lines, each filled by one width-W word. Both gadgets therefore end with a full line, contributing one and two line breaks respectively, so the final word has line index

$$1 + N + \sum_{i=0}^{N-1} x_i.$$

The width sequence is an AC^0 encoding of the input bits, and an AC^0 comparison of the returned line index with a hardwired threshold decides MAJORITY. Since MAJORITY generates TC^0 under AC^0 Turing reductions (Chandra et al., 1984), the wrapping function is TC^0-hard.

We now transfer this classification to the rendered image. To compute Render_W with an oracle for Wrap_W, first obtain each $a_i = |s_i|$ by a constant-size lookup and query the word positions (ℓ_i, ζ_i). Each output pixel is then a fixed glyph lookup on the characters of s_i, followed by an OR over the words, all in AC^0. Since $\mathrm{Wrap}_W \in \mathsf{TC}^0$, this reduction places Render_W in TC^0.

For the reverse reduction, nontriviality provides two characters $u, v \in \Sigma$ whose glyph rasters differ at a fixed pixel. For each word index $i \in \{1, \ldots, m\}$, form two rendering inputs with the same paragraph indicators. Both inputs represent word j by u^{a_j} for every $j \in \{1, \ldots, m\}$. In the second input, replace the first u of the i-th word by v.

The two inputs have identical layouts because the font is monospaced. Comparing their bitmaps at that pixel offset in every character cell gives a one-hot encoding of the starting cell (ℓ_i, ζ_i). An AC^0 circuit converts this encoding to binary. All $O(m)$ pairs are queried in parallel with polynomially many oracle gates. This gives an AC^0 Turing reduction from Wrap_W to Render_W. Since Wrap_W is TC^0-hard, Render_W is TC^0-hard as well. □

D EXPERIMENTAL DETAILS

D.1 OVERALL SETUP

Backbones. For line tracing, we train a 9-million-parameter multimodal diffusion transformer (MMDiT) from random initialization, following the `FLUX.2` architecture (Black Forest Labs, 2026). It has three joint-attention blocks and three single-stream blocks, each with four attention heads and head dimension 64. Other runs fine-tune pretrained backbones with LoRA adapters (rank and alpha 32, no dropout). Long multiplication, long text rendering, and the controlled comparison of §4.3 use `FLUX.2-klein-base-4B` (Black Forest Labs, 2026). The large-model comparison in §4.3 instead uses `Qwen-Image-Edit-2511` (Qwen Team, 2025).

Endpoint-prediction conditioning. All backbones encode images into VAE latents and pack each 2×2 latent patch into one image token. In the second training pass and at each sampling step after the first, SELFCONDFM concatenates the endpoint-prediction tokens with the noisy-target tokens and any conditioning-image tokens before the transformer blocks. The `FLUX.2` backbones append the endpoint-prediction tokens with a distinct positional encoding offset, and `Qwen-Image-Edit` places them after the target-image tokens. No attention mask separates the image-token blocks. Both placements follow the backbones' own design for image-conditioned generation. Thus self-conditioning changes the input sequence but not the transformer architecture.

Training-FLOP accounting. We match training compute by cumulative FLOPs, the total number of floating-point operations a run performs. We estimate it as the forward-and-backward FLOPs per micro-batch, averaged over 100 micro-batches, multiplied by the number of micro-batches processed across all devices. Each measurement covers the complete training computation, including VAE encoding, the second SELFCONDFM training pass, and the longer self-conditioned token sequence, so the extra cost of self-conditioning is fully counted. The plotted budgets in Figure 1 are 4.50×10^{18} FLOPs for line tracing, 2.32×10^{19} for long multiplication, and 4.70×10^{19} for long text rendering.

D.2 SETUP AND EVALUATION FOR THE TASKS IN §4.2

Shared configuration. All training runs are conducted on four NVIDIA A100 80GB GPUs, using AdamW, bfloat16 precision, and a constant learning-rate schedule with $1{,}000$ warmup steps. Within each task, the two methods share the data, optimizer, noise schedule, and evaluation procedure. The SELFCONDFM runs use the two-pass training scheme of §3.2.

Line tracing. The training set contains 10^6 random instances with $m = 50$. We train with a global batch size of 32 and a learning rate of 2×10^{-3}. After FLOP alignment, the plotted curves end at step 60k for VANILLAFM and step 53k for SELFCONDFM.

We evaluate checkpoints on a held-out set of 20 instances using 100 NFE. The evaluator samples interior locations on the five expected segments of each stage and requires their local average RGB values to match the ground-truth rendering within a fixed tolerance. Our evaluation metric is the *longest correct prefix* (LCP), defined for each instance as the length of the longest prefix of stages for which all five segments match.

Long multiplication. The training set contains $2{\times}10^5$ problems with five- to seven-digit operands, where the number of digits of each operand is drawn independently and uniformly from $\{5, 6, 7\}$. Each operand is then sampled uniformly at random from the integers with the chosen number of digits, so it contains no leading zeros. We use a global batch size of 8 and a learning rate of 10^{-4}, training SELFCONDFM for five epochs and VANILLAFM for 15 epochs to cover a comparable FLOP range.

We evaluate checkpoints on a 100-instance test split drawn from the same distribution as the training set, using 50 NFE. We use `Qwen3.8-Flash` (Qwen Team, 2026) as the judge, prompting it to compare the generated and reference worksheets. A sample counts as correct only when the two worksheets agree throughout, from the final product to every intermediate row. The full judge prompt is given in Figure 4.

> You are evaluating a visual reasoning model on a column-multiplication task. You will see a problem setup image (the two numbers to multiply, written in column form), the correct solution image (the fully worked-out column multiplication with the correct final product), and the model's predicted solution image.
>
> Decide whether the predicted solution is correct: it must correspond to the same problem setup as the correct solution, the final product it shows must equal the correct product, and the intermediate steps, including the carries, must match the correct solution as well. If the predicted solution is blank, truncated, or unreadable, answer '[NO]'. Provide a brief explanation, then answer '[YES]' if the predicted solution is correct and '[NO]' if it is not.

Figure 4: Judge prompt for the long-multiplication evaluation.

Long text rendering. The training set contains 3×10^5 text samples as conditions, with their 384×384 monospaced renderings as targets. The corpus is drawn from the TinyStories dataset (Eldan & Li, 2023). During preprocessing, we truncate each text to the longest prefix that fits on the canvas. We use a global batch size of 16 and a learning rate of 10^{-4}, training VANILLAFM for 12 epochs and SELFCONDFM for four epochs.

We evaluate 100 held-out instances using 50 NFE, parsing the text from the reference and generated images with `Qwen3.8-Flash` (Qwen Team, 2026), prompted to preserve all line breaks and whitespace. The rate of perfect rendering is the fraction of instances whose two parsed texts match exactly.

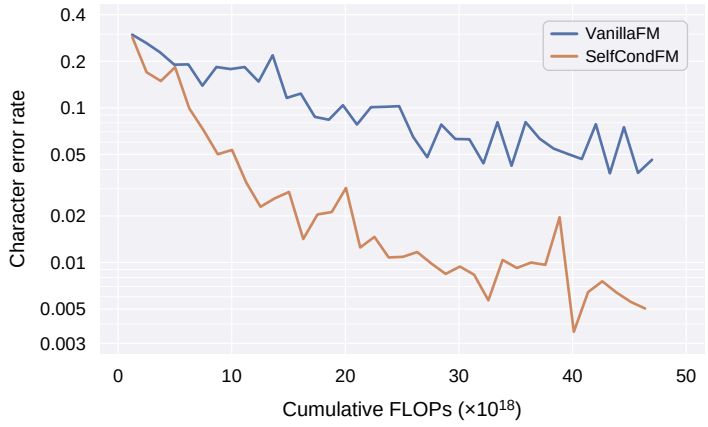


Figure 5: Character error rate (CER) of parsed text from the long text rendering evaluation, against cumulative training FLOPs. SELFCONDFM keeps improving throughout training, whereas VANILLAFM plateaus nearly an order of magnitude above the final SELFCONDFM value.

Beyond exact matching, we also compute the character error rate (CER) between the two parsed texts (Figure 5), defined as $\mathrm{CER} = d_{\mathrm{Lev}}(r,\hat{r})/|r|$, where r and $\hat{r}$ are the parsed text of the reference and generated images, respectively. The Levenshtein distance $d_{\mathrm{Lev}}(r,\hat{r})$ is the minimum number of character edits required to transform r into $\hat{r}$, where each edit inserts, deletes, or substitutes one character and has unit cost. SELFCONDFM ends at CER 0.005, while VANILLAFM plateaus between 0.03 and 0.1 and stands at 0.046 at the matched budget of 4.70×10^{19} FLOPs.

D.2.1 MULTI-SEED EVALUATION

We further train VANILLAFM and SELFCONDFM with 3 different seeds (i.e., 2 seeds in addition to the one used in Figure 1) on line tracing and long multiplication. We evaluate line tracing on the same 20-instance held-out set as the single-seed evaluation at 100 NFE, and long multiplication on the same 100-instance held-out set as its single-seed evaluation at 50 NFE. Due to limited computational

resources, we train long multiplication with a smaller compute budget and do not run multi-seed experiments for long text rendering.

Figures 6 and 7 show the mean and the one-standard-deviation band of the evaluation curves of each method against training FLOPs. For line tracing, at the training-FLOP-matched checkpoints (SELFCONDFM@53k vs. VANILLAFM@60k), SELFCONDFM saturates at a mean LCP of 50.0 ± 0.0, while VANILLAFM reaches only 5.6 ± 2.7. For long multiplication, at its smaller matched budget (SELFCONDFM@50k vs. VANILLAFM@125k), SELFCONDFM attains $(90.0 \pm 5.3)\%$ accuracy while VANILLAFM reaches $(35.0 \pm 8.9)\%$. These multi-seed results thus confirm that SELFCONDFM outperforms VANILLAFM on line tracing and long multiplication, consistent with the single-seed evaluations in Figure 1.

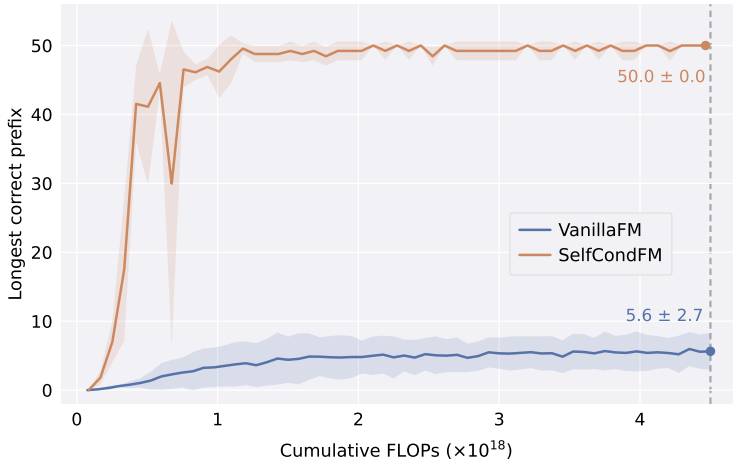


Figure 6: Training curves of line tracing with 3 different seeds, plotting mean LCP on the 20-instance held-out set at 100 NFE against training FLOPs. For each method, the curve shows the mean over the 3 runs, and the shaded band shows the mean $\pm$ one standard deviation. SELFCONDFM saturates at 50.0 ± 0.0, while VANILLAFM reaches 5.6 ± 2.7 at the matched budget.

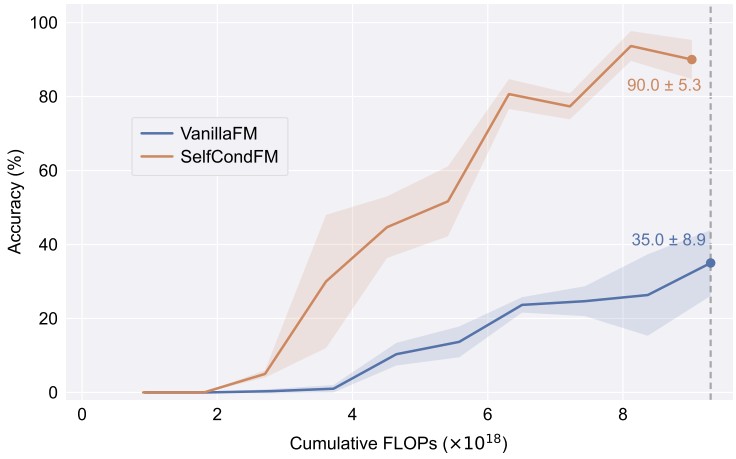


Figure 7: Training curves of long multiplication with 3 different seeds, plotting the mean accuracy on the held-out set under 50 NFE against training FLOPs. As in Figure 6, each curve and its shaded band show the mean and the mean $\pm$ one standard deviation over the 3 runs. VANILLAFM reaches $(35.0 \pm 8.9)\%$ while SELFCONDFM is higher at $(90.0 \pm 5.3)\%$ under the fixed budget.

D.2.2 GENERATED EXAMPLES

Figures 8 to 10 present selected held-out examples on which VANILLAFM fails while SELFCONDFM succeeds, with annotations highlighting the errors made by VANILLAFM. They are generated by the checkpoints at the final evaluation point of Figure 1, namely VANILLAFM@60k and SELFCONDFM@53k for line tracing, VANILLAFM@312.5k and SELFCONDFM@125k for long multiplication, and VANILLAFM@190k and SELFCONDFM@74k for long text rendering.

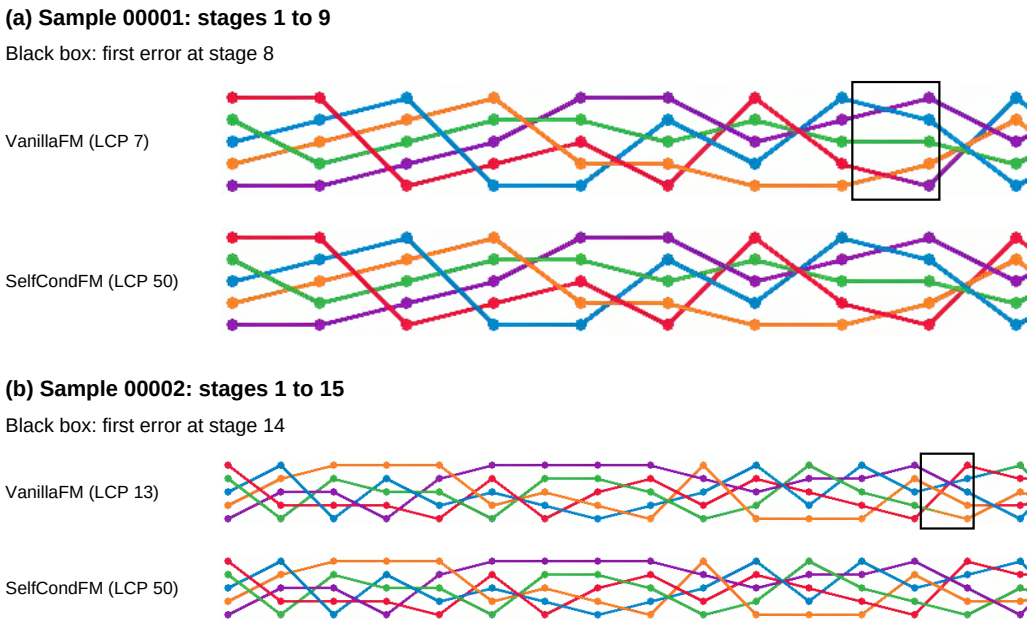
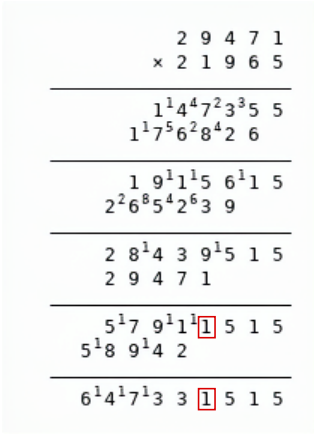


Figure 8: Line-tracing examples. For visualization, each trajectory is truncated shortly after the first error of VANILLAFM; the full trajectory contains 50 stages. VANILLAFM first fails at stages 8 and 14 in (a) and (b), respectively, by incorrectly switching colors at line crossings. Black boxes mark the errors. In contrast, SELFCONDFM preserves each line's color throughout all 50 stages in both examples (LCP $= 50$).

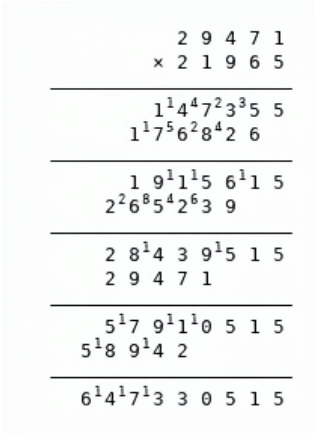
(a) 29471 x 21965

VanillaFM



2 incorrect digits

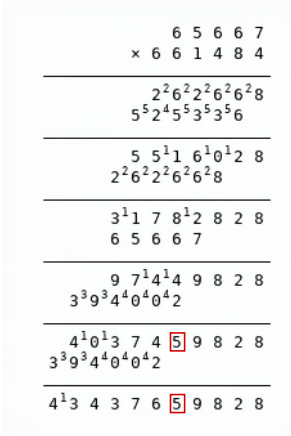
SelfCondFM



Correct

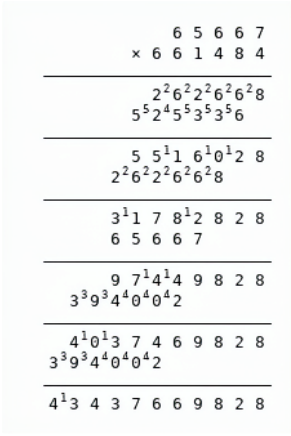
(b) 65667 x 661484

VanillaFM



2 incorrect digits

SelfCondFM

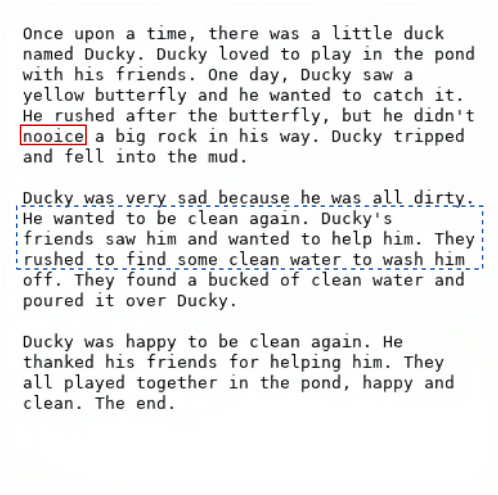


Correct

Figure 9: Long-multiplication examples. VANILLAFM makes accumulation errors in both cases, resulting in incorrect final answers, with the erroneous digits highlighted in red boxes. In contrast, SELFCONDFM produces the correct product for both examples.

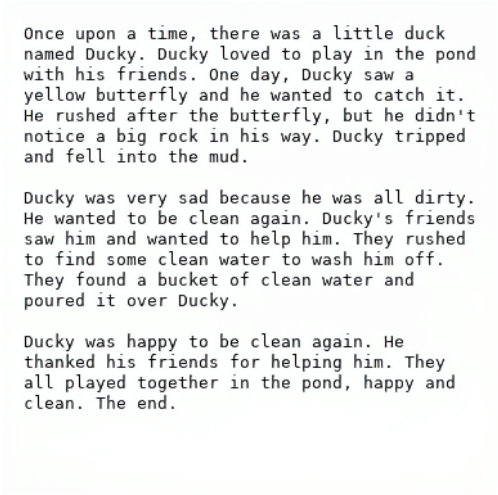
(a) Sample 00007_000082

VanillaFM



CER = 0.0106

SelfCondFM



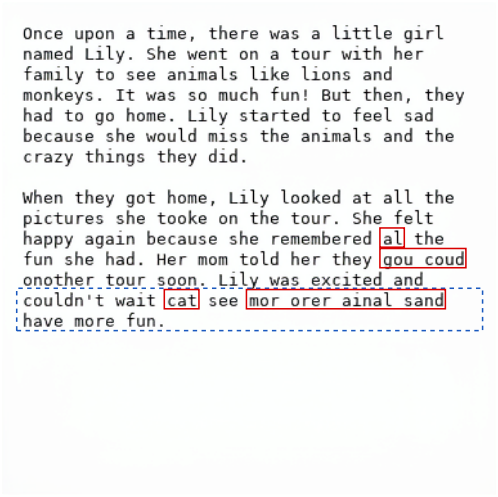
CER = 0.0000

Red boxes: text errors

Blue dashed boxes: line-break and whitespace errors

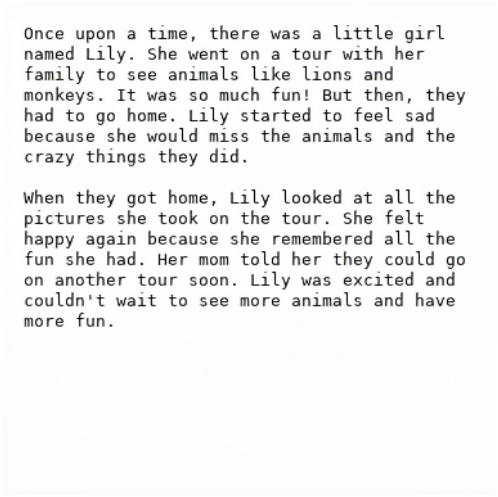
(b) Sample 00018_000043

VanillaFM



CER = 0.0410

SelfCondFM



CER = 0.0000

Red boxes: text errors

Blue dashed boxes: line-break and whitespace errors

Figure 10: Long text rendering examples. VANILLAFM produces text errors as well as line-break and whitespace errors, yielding CERs of 0.0106 and 0.0410 in (a) and (b), respectively. Red boxes highlight text errors, and blue dashed boxes indicate line-break and whitespace errors. In contrast, SELFCONDFM renders both passages correctly with zero CER.

D.3 NFE Scaling

For Figure 3, we evaluate the training-FLOP-matched line-tracing checkpoints of Appendix D.2, VanillaFM@60k and SelfCondFM@53k, on 200 held-out instances with $m = 50$. The NFE grid covers every integer from 1 to 50, and the two methods share the same initial noise for each instance at every NFE. The LCP is computed with the evaluator described in Appendix D.2.

Beyond the NFE-matched comparison of Figure 3, we also plot the mean LCP against cumulative inference FLOPs in Figure 11.

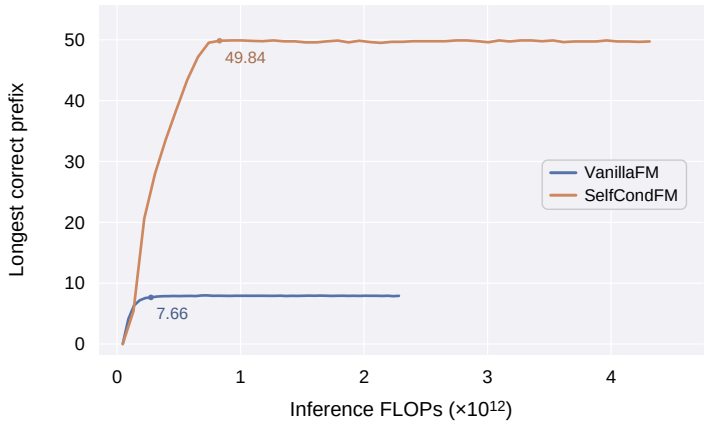


Figure 11: Mean LCP in the NFE scaling comparison of Figure 3, plotted against cumulative inference FLOPs. Both methods have zero mean LCP at NFE $= 1$. SelfCondFM reaches 49.84 at 8.29×10^{11} FLOPs. VanillaFM reaches 7.66 at 2.74×10^{11} FLOPs, ending below 8.0 up to 2.28×10^{12} FLOPs (which corresponds to NFE $= 50$).

D.4 Setup and Results for Structured Visual-Reasoning Tasks

Datasets and evaluators. We prepare the training data with the pipeline of Dai et al. (2026) for VSP-Super, Maze, TSP, and Sudoku, and with that of He et al. (2025) for Jigsaw. The evaluation data and protocol for all five tasks follow He et al. (2025).

Controlled experiments. In the controlled experiments of §4.3, for each of VSP-Super, Maze, TSP, Sudoku, and Jigsaw, we train separate `FLUX.2-klein-base-4B` LoRA adapters under the shared configuration of the tasks in §4.2, with a global batch size of 16 and a learning rate of 10^{-4}. Checkpoints are compared at matched cumulative training FLOPs, at which SelfCondFM has been trained for 16k steps, and all results use 50 NFE.

Large-scale comparisons. Following the setup of Dai et al. (2026), we train two kinds of SelfCondFM models. For task-specific training, we train one `Qwen-Image-Edit-2511` LoRA adapter per task. For unified training, a single adapter is trained on the union of the VSP-Super, Maze, TSP, and Sudoku data. All adapters are trained for five epochs to ensure convergence.

D.4.1 Jigsaw Results

Since EndoCoT (Dai et al., 2026) does not evaluate Jigsaw, we compare our large-model result on Jigsaw with the baselines reported by He et al. (2025). Our SelfCondFM adapter attains the best average accuracy, 96.0% versus 92.3% for DiffThinker, with the largest gain at the hardest level; see Table 3.

Table 3: Jigsaw accuracy (%) at difficulty levels 2 to 4. "Avg." is the unweighted mean over the three levels. The best results are in bold and the second best are underlined.

Method	2	3	4	Avg.
Gemini-3-Flash	71	16	0	29.0
GPT-5	30	0	0	10.0
Qwen3-VL-8B	7	0	0	2.3
Qwen3-VL-8B (SFT)	95	56	9	53.3
Qwen3-VL-8B (GRPO)	6	0	0	2.0
DiffThinker	**99**	<u>98</u>	<u>80</u>	<u>92.3</u>
Ours	<u>98</u>	**100**	**90**	**96.0**

D.4.2 GENERATED EXAMPLES

To illustrate the qualitative differences between VANILLAFM and SELFCONDFM on the five structured visual-reasoning tasks, we select two held-out instances per task from the controlled comparison in Table 1 on which VANILLAFM fails while SELFCONDFM succeeds. The examples are shown in Figures 12 to 16, with annotations highlighting the errors made by VANILLAFM.

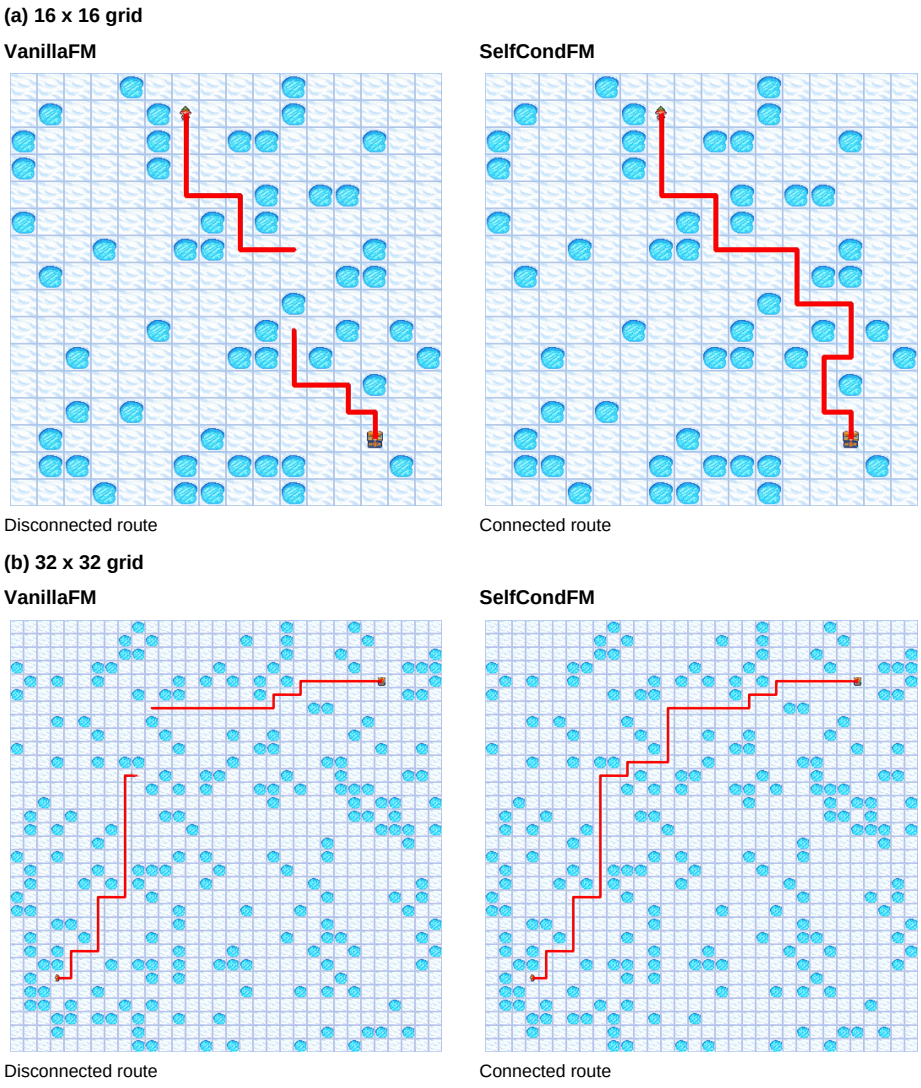


Figure 12: VSP-Super examples on 16-by-16 and 32-by-32 grids. The VANILLAFM routes contain disconnected segments. In contrast, the SELFCONDFM routes correctly connect the endpoints.

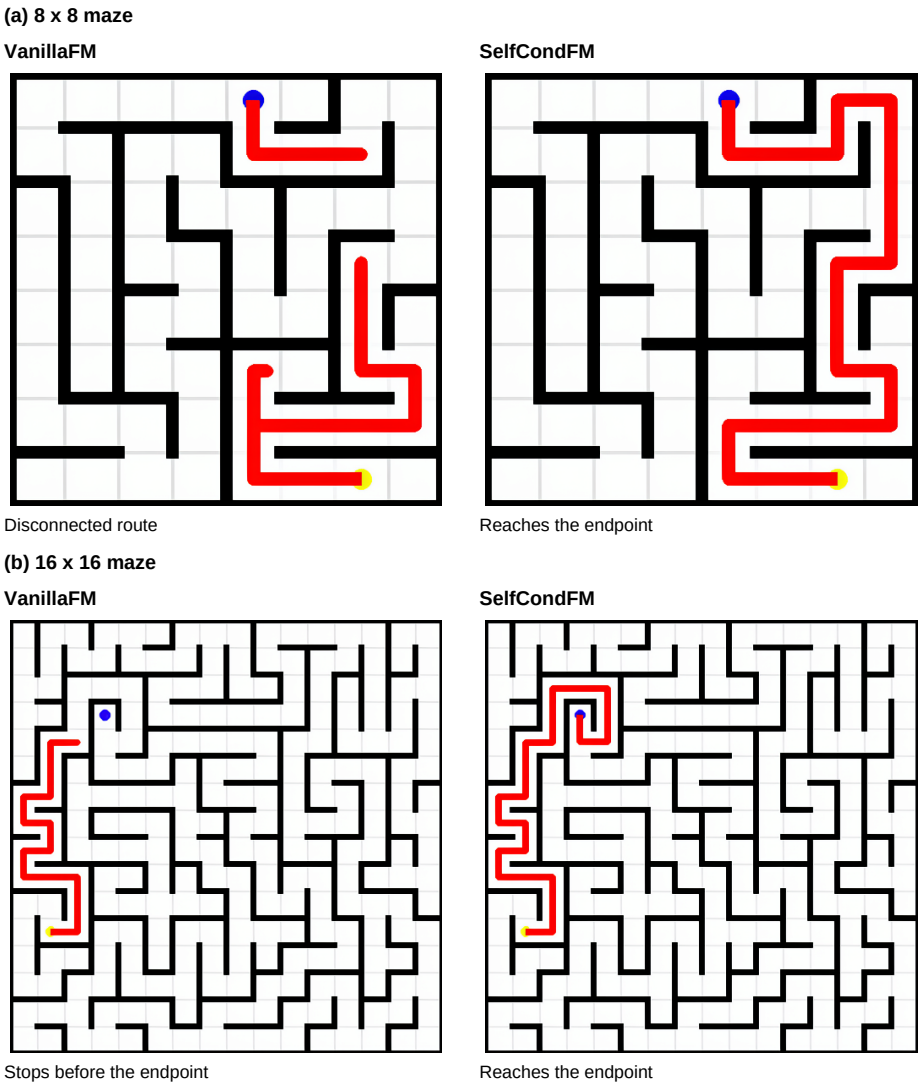


Figure 13: Maze examples on 8-by-8 and 16-by-16 grids. In (a), VANILLAFM leaves separate path fragments; in (b), its route stops before reaching the blue endpoint. In contrast, SELFCONDFM completes both routes.

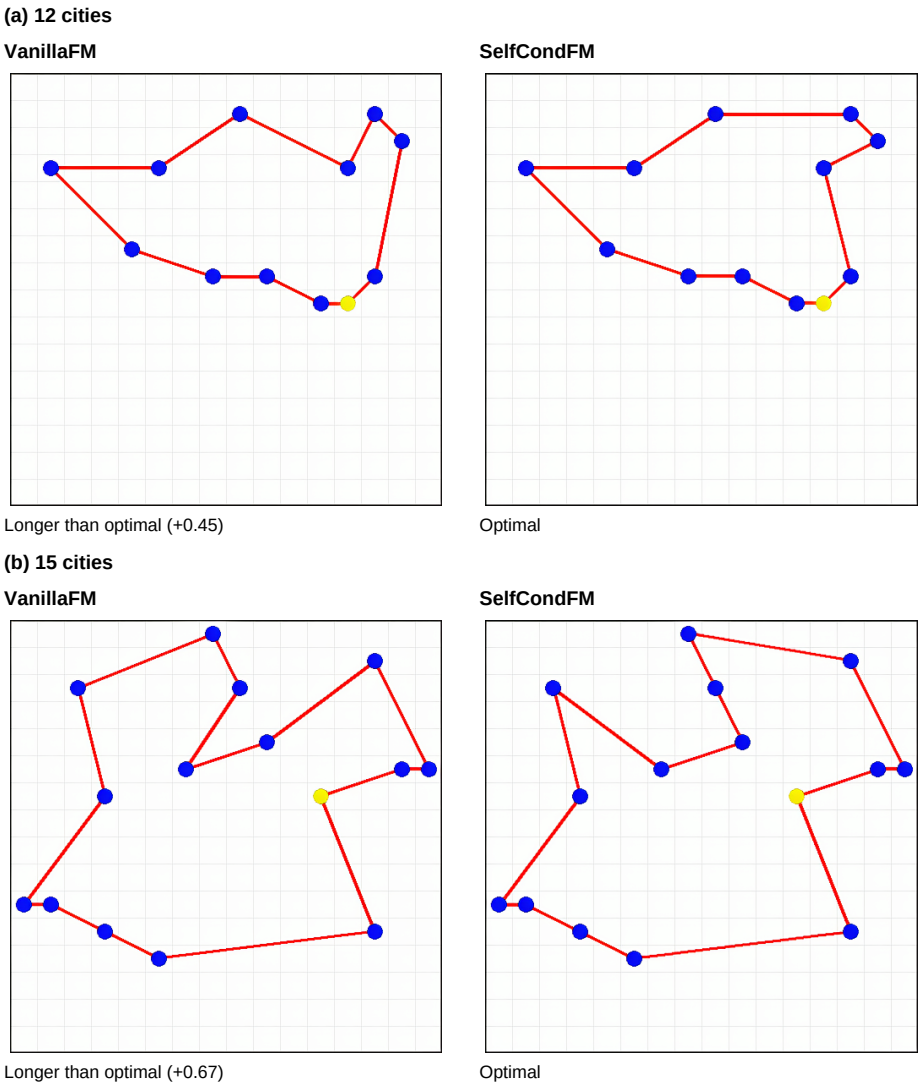


Figure 14: TSP examples with 12 and 15 cities. VANILLAFM fails to find the optimal tour in both cases, yielding excess tour lengths of +0.45 and +0.67, respectively. In contrast, SELFCONDFM produces the optimal solution for both instances.

(a) 35 given digits

VanillaFM

1	2	6	7	8	3	9	4	5
7	9	3	2	5	4	6	1	8
8	4	5	9	6	1	2	2	3
4	1	2	3	9	8	7	5	6
6	8	7	4	1	5	3	9	2
5	3	9	6	2	7	4	8	1
2	7	1	5	4	6	8	3	9
9	6	8	1	3	2	5	7	4
3	5	4	8	7	9	1	6	2

5 incorrect cells

SelfCondFM

1	2	6	7	8	3	9	4	5
7	9	3	2	5	4	6	1	8
8	4	5	9	6	1	7	2	3
4	1	7	3	9	8	2	5	6
6	8	2	4	1	5	3	9	7
5	3	9	6	2	7	4	8	1
2	7	1	5	4	6	8	3	9
9	6	8	1	3	2	5	7	4
3	5	4	8	7	9	1	6	2

Correct

(b) 45 given digits

VanillaFM

1	7	3	4	6	2	5	9	8
6	9	2	5	7	8	4	1	3
5	4	8	3	1	9	6	7	2
7	8	1	9	2	4	3	5	6
9	2	4	1	3	5	1	8	7
3	5	1	7	8	6	2	4	9
4	1	7	2	9	3	8	6	5
8	3	9	6	5	1	7	2	4
2	6	5	8	4	7	9	3	1

5 incorrect cells

SelfCondFM

1	7	3	4	6	2	5	9	8
6	9	2	5	7	8	4	1	3
5	4	8	3	1	9	6	7	2
7	8	1	9	2	4	3	5	6
9	2	4	6	3	5	1	8	7
3	5	6	7	8	1	2	4	9
4	1	7	2	9	3	8	6	5
8	3	9	1	5	6	7	2	4
2	6	5	8	4	7	9	3	1

Correct

Figure 15: Sudoku examples with 35 and 45 given digits. VANILLAFM produces incorrect solutions with explicit Sudoku constraint violations: row 3 in (a) contains two 2s, while row 5 in (b) contains two 1s. The red boxes highlight these errors. In contrast, SELFCONDFM correctly solves both puzzles.

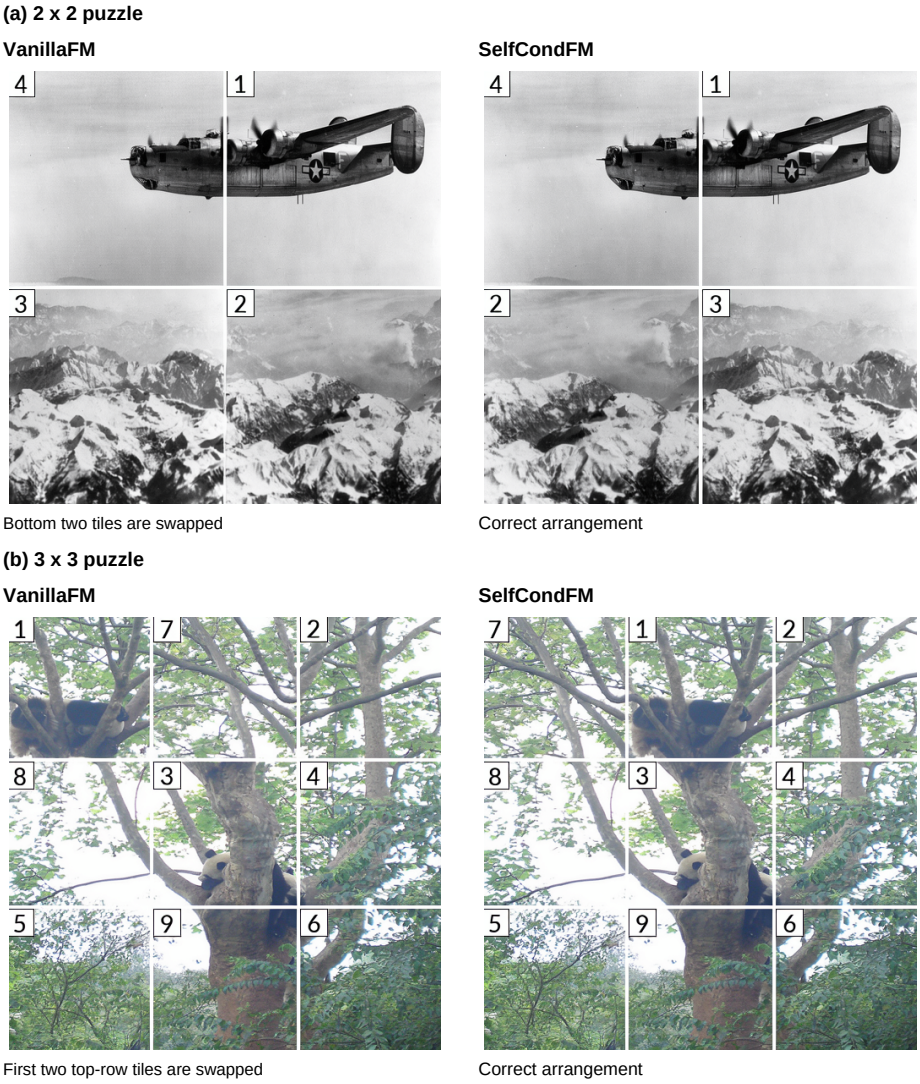


Figure 16: Jigsaw examples on 2-by-2 and 3-by-3 tile grids. VANILLAFM fails in both examples: it swaps the two bottom tiles of (a), giving the order 3, 2 instead of 2, 3, and swaps the first two top-row tiles of (b), giving 1, 7 instead of 7, 1. SELFCONDFM matches the reference permutation in both examples. Tile identifiers are part of the original images.

D.5 Additional Empirical Analyses

D.5.1 Empirical Lipschitz Estimation

Theorem 1 assumes that the model u has a bounded Lipschitz constant. To empirically support this assumption, we estimate the Lipschitz constant of a VanillaFM line-tracing model in the joint ℓ_∞ norm on (x, ctx) used in §2.2. The checkpoint is separately trained on approximately 3×10^5 mixed-length examples, using about 2×10^{17} training FLOPs.

For a state x_i visited at time t_i, we use projected gradient ascent to maximize

$$\frac{\|u(x_i + \Delta x, t_i; \mathrm{ctx} + \Delta \mathrm{ctx}) - u(x_i, t_i; \mathrm{ctx})\|_\infty}{\max\{\|\Delta x\|_\infty, \|\Delta \mathrm{ctx}\|_\infty\}}.$$

We jointly perturb the noisy state and the condition latent with radii 0.003, 0.01, 0.03, 0.1, and 0.3, and cover every state of a 50-step inference trajectory. For each problem length $m = 1, \ldots, 50$, we estimate L on 30 held-out trajectories and report the maximum estimate over all tested states, radii, and trajectories.

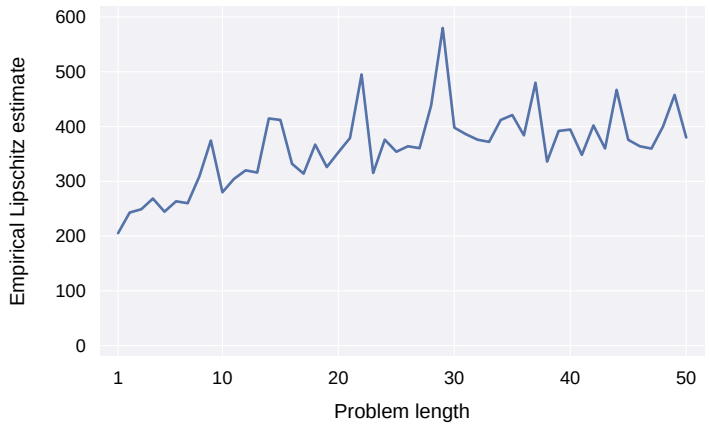


Figure 17: Maximum joint ℓ_∞ empirical Lipschitz estimate among 30 held-out trajectories at each problem length.

As shown in Figure 17, the per-length maximum rises for small m and levels off around 400 for $m \geq 30$, with an overall maximum of 580. The largest observed velocity norm is below 10. The estimates thus show no continued growth with m over the tested range.

Remark. These estimates are not certified upper bounds. Nevertheless, estimating Lipschitz constants via adversarial search is common practice (Yang et al., 2020; Achour et al., 2022), and the absence of continued growth in the estimated L with m still provides empirical support for the bounded-Lipschitz regime assumed in §3.1.

D.5.2 Attention Allocation

We measure how the first joint-attention layer of the line-tracing SelfCondFM checkpoint allocates attention from queries at the x_i token positions to the keys from three input sources at sampling step i: the noisy state x_i, the conditioning image w, and the endpoint prediction z_i. The experiment uses 50 test instances with $m = 50$ and 50 NFE. At every step after the first, we normalize the attention mass over the three sources and average it over heads and instances.

As shown in Figure 18, the attention share of the endpoint prediction stays between 20.0% and 31.3% and declines near-monotonically along the sampling trajectory. Thus the model relies on z_i throughout sampling and uses it most heavily at early steps, where the noisy state is less informative.

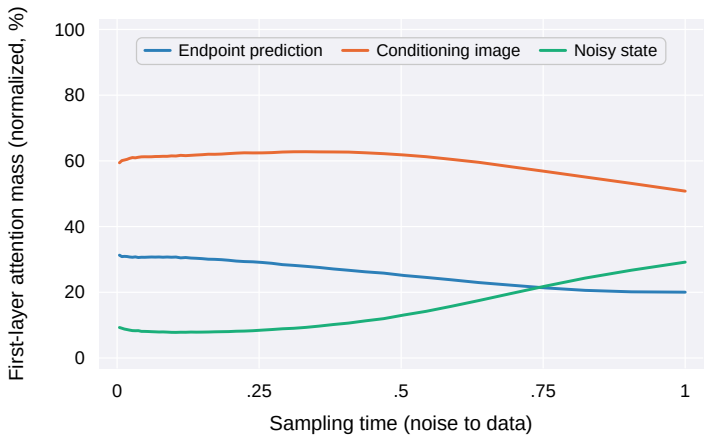


Figure 18: Normalized first-layer attention mass to each input source. Endpoint prediction maintains a nontrivial share throughout the trajectory, declining near-monotonically from 31.3% to 20.0%, thus peaking in the early, noise-dominated regime.

D.5.3 Parallel and Revisable Reasoning

By inspecting the sampling trajectories of the long-multiplication checkpoints, we observe two properties of visual generative reasoning with flow matching that SelfCondFM preserves: parallel local computation and in-place error correction, which are usually not possible in autoregressive reasoning. The following paragraphs provide empirical evidence for each.

Parallel local computation. For each generated worksheet, we score every shifted partial product AB_i10^i against the reference, repeating the evaluation over every integer NFE from 1 to 50. As shown in Figure 19, the partial-product accuracy of both methods stays above 98% at every NFE, even at NFE $= 1$. This shows that even a single model evaluation can produce multiple correct local results simultaneously within the same worksheet. By contrast, common autoregressive reasoning methods, such as CoT, generate reasoning steps sequentially and do not produce multiple local results in parallel.

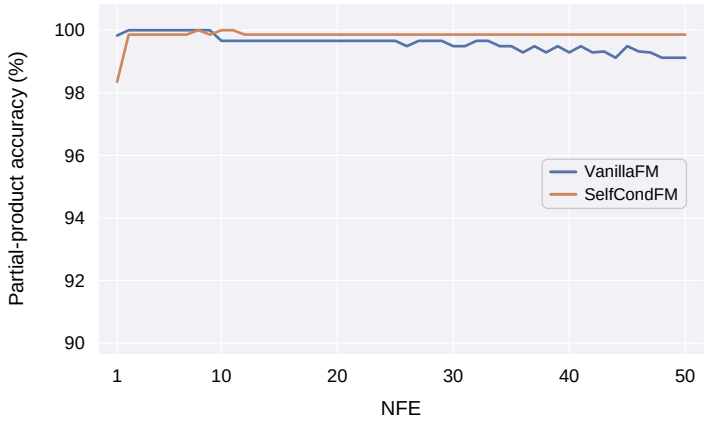


Figure 19: Parallel local computation. Partial-product accuracy on long multiplication at every NFE from 1 to 50: both methods keep nearly all partial products correct within a single generated worksheet.

In-place error correction. For pairs of problems (w, w') with different answers, we sample with 50 NFE, conditioning on w for the first k steps and swapping to w' for the rest. We repeat this for every k from 1 to 50, and judge each output against the reference worksheet of w'.

As shown in Figure 20, SELFCONDFM keeps the match-w' accuracy around 90% whenever the swap falls in the first half of the trajectory, and the accuracy falls to zero only once k exceeds 35. VANILLAFM also corrects early swaps in place, but its accuracy is only 65% at $k = 1$ and decays to zero by $k = 32$.

Both methods therefore revise a wrong mid-trajectory generation in place, with SELFCONDFM performing particularly well in the first half of the trajectory. Autoregressive reasoning, in contrast, retains the reasoning steps once they are generated and cannot correct them in place.

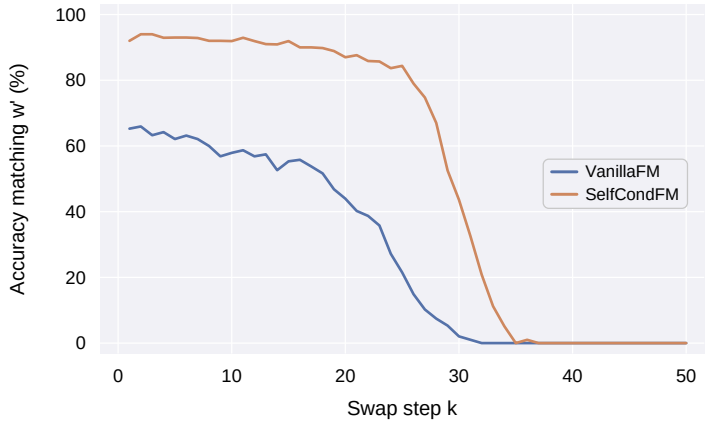


Figure 20: In-place error correction on long multiplication. The condition changes from w to w' after step k, and the curves show the accuracy of matching w' as k varies: the output of SELFCONDFM is redirected in place to the new problem when the swap falls in the first half of the trajectory, whereas VANILLAFM follows the new problem only partially.

E Additional Related Work

E.1 Generative Visual Reasoning

Internal visual intermediates. Several approaches let the model produce its own visual intermediates during multimodal reasoning, in the form of generated images, latent visual states, or learned visual representations (Xu et al., 2026; Yang et al., 2026; Zhang et al., 2025; Qin et al., 2025; Wang et al., 2026a; Gu et al., 2026; Li et al., 2026; Cheng et al., 2026). These intermediates give the model a richer workspace for reasoning than text alone.

Reasoning in video generation. Video generation offers another medium for reasoning, representing evolving intermediate states as a sequence of frames (Wiedemer et al., 2025; Tong et al., 2026). Recent works further suggest that video diffusion models (Ho et al., 2022; Xing et al., 2024) reason across denoising steps: they explore candidate solutions, correct earlier mistakes, and commit early to high-level plans (Wang et al., 2026b; Newman et al., 2026).

E.2 Computational Expressiveness

Transformers and circuit complexity. The expressive power of transformers (Vaswani et al., 2017) has been studied extensively through circuit complexity, formal languages, and logic (Strobl et al., 2024). Depending on the attention mechanism and numerical precision, constant-depth transformers are upper-bounded by AC^0 or TC^0 (Hao et al., 2022; Merrill et al., 2022; Merrill & Sabharwal, 2023; Chiang, 2024), and such circuit bounds extend to modern components such as rotary positional embeddings (Chen et al., 2025). Chiang et al. (2023) further tighten both the upper and lower bounds through a logical characterization.

Sequential computation beyond a single evaluation. Repeated model calls can lift computation beyond these single-evaluation bounds. With chain-of-thought (Wei et al., 2022), a constant-depth transformer writes its intermediate results into the autoregressive context and can thereby solve inherently serial problems (Li et al., 2024). Amiri et al. (2025) complement this with lower bounds on the number of chain-of-thought steps that hard-attention transformers need for various algorithmic problems.