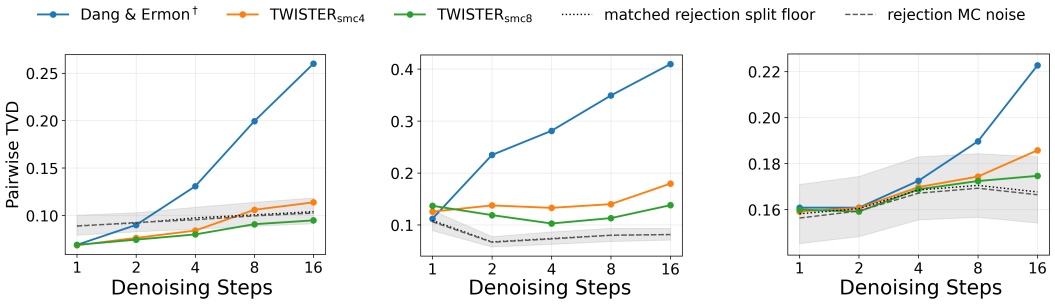
# TWIST, DON'T TILT: TRAJECTORY-EXACT CONSTRAINED DECODING FOR MASKED DIFFUSION MODELS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Constrained decoding for Masked Diffusion Language Models (MDLMs) aims to ensure that generated outputs satisfy a specified structure or syntax constraint. MDLMs generate outputs by repeatedly unmasking masked positions present in their current state. Recent strategies for constrained decoding constrain the model's per-step mean-field posterior (which factorizes over masked positions) by enforcing the desired constraint with an automaton. The resulting chain-structured factor graph allows exact constrained sampling via dynamic programming. However, despite each draw being exact and constraint-satisfying, we prove that their composition, in general, tilts away from the model's relative probabilities over valid trajectories, thus leading to trajectory bias. We derive an exact expression for this bias as a product of ratios measuring how valid continuation mass changes when the denoiser is reconditioned, and characterize when the bias vanishes. We then correct the bias by introducing TWISTER, the first automaton-twisted Sequential Monte Carlo decoder for MDLMs, using the step-exact decoder as the proposal. We show that for regular language constraints, the Feynman–Kac correction is exactly computable, with the twists obtained efficiently using quantities pre-computed for step-exact sampling. We prove that the resulting Feynman–Kac model targets the unbiased Doob $h$-transformed path law conditioned on constraint satisfaction.




**(a)** OWT-130M **(b)** DREAMCODER-7B-INST **(c)** DREAM-7B-BASE

**Figure 1:** Trajectory bias in constrained decoding of MDLMs captured by the Total Variation Distance (TVD) between the Doob $h$-transformed native path law (rejection sampling) and the step-exact automaton-constrained distribution (Dang & Ermon (2026), TWISTER$_{smc4}$ and TWISTER$_{smc8}$) across denoising steps. For "a.*".

## 1 INTRODUCTION

Masked Diffusion Language Models (MDLMs) (Sahoo et al., 2024; Nie et al., 2025; Xie et al., 2025; Ye et al., 2025) are increasingly used for generating text, code, and other data, that may require their outputs to adhere to a specified format structure (syntax), e.g., a JSON schema.

Unlike autoregressive Large Language Models (LLMs), MDLMs decode by repeatedly denoising a masked input; they first predict mask-free token-categoricals over the masked positions, and then retain only a scheduled subset of these predictions. Therefore, the generation order depends on the sequence of these scheduled subsets of positions and is not a left-to-right prefix like in LLMs. This creates a different kind of constrained decoding problem. Suresh et al. (2025) introduced DINGO, an MDLM algorithm for regular languages, which used maximum a posteriori (MAP) decoding over



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.

the chain-structured factorization of the model's automaton-constrained posterior. While DINGO is mode-seeking, Dang & Ermon (2026) proposed an algorithm for instead sampling (exactly) from the automaton-constrained posterior at each step using Forward-Filtering Backward-Sampling (FFBS) (Carter & Kohn, 1994).

We show that sampling locally from the automaton-constrained posterior at each step, although exact, is still biased when composed across steps, thus deviating from the globally constrained distribution. The bias arises because the denoising process only retains a scheduled subset of the sampled tokens and discards the rest when producing the next intermediate state. The resulting *clamped-partition sum* is the total constraint-valid probability mass obtained by fixing (or clamping) the retained predictions to the successor state and marginalizing the discarded ones under the current-state-conditioned MDLM. At the next step, the same positions (discarded in the previous step) are reevaluated under the successor-state-conditioned MDLM, producing a new local-partition sum. The ratios of the clamped-partition sum to the local-partition sum tilt the resulting path law relative to the globally constrained one, which we model via the Doob $h$-transform (Doob, 1957) of the native path law, conditioned on constraint-satisfaction. The tilt manifests as *trajectory bias*. Figure 1 shows the trajectory bias when sampling locally from the automaton-constrained posterior using a step-exact decoder, relative to the native decoder conditioned on constraint satisfaction, for different models.

Unlike locally constrained LLM decoding, which ignores future valid mass (Park et al., 2024; Dang et al., 2026), step-exact MDLM decoding accounts for this mass exactly, but under a denoiser frozen at the current state. When these exact local steps are composed, reconditioning the denoiser can change the valid mass and bias the resulting distribution. We correct the bias by introducing TWISTER, which uses a *Feynman–Kac correction* with the local and clamped-partition sum ratios as the incremental potentials. The correction requires quantities already computed by FFBS of the step-exact sampler proposed by Dang & Ermon (2026), making it exactly and efficiently computable.

## 2 BACKGROUND

In this section, we introduce notation used in the rest of the paper, as well as background on MDLMs and twisted Sequential Monte Carlo. We use $[N]$ to denote $\{1, 2, \ldots, N\}$, the set of natural numbers up to $N$; and $[\![\cdot]\!]$ to denote the standard indicator function. For two $N$-length sequences $\mathbf{x}$ and $\mathbf{x}'$, we write $[\![\mathbf{x} \equiv_r \mathbf{x}']\!]$ if $\mathbf{x}(i) = \mathbf{x}'(i)$ for every $i \in r \subseteq [N]$. For bounded products such as $\prod_{t<T}$, we always assume $t$ starts from 0. Unnormalized quantities are generally denoted by a tilde, e.g., $\widetilde{\pi}$.

**Deterministic Finite Automata and Regular Languages**. A deterministic finite automaton (DFA) over a finite alphabet $\Sigma$ is a tuple: $\mathcal{A} \triangleq \langle \Sigma, \mathcal{Q}, q_0, \mathcal{F}, \delta \rangle$, where $\mathcal{Q}$ is a finite set of states, $q_0 \in \mathcal{Q}$ is the initial state, $\mathcal{F} \subseteq \mathcal{Q}$ is the set of accepting states, and $\delta : \mathcal{Q} \times \Sigma \to \mathcal{Q}$ is the transition function. A word $\boldsymbol{w}$ is accepted by a DFA if the DFA ends in an accepting state after consuming $\boldsymbol{w}$, i.e., $\delta^*(q_0, \boldsymbol{w}) \in \mathcal{F}$[‡].

**Definition 2.1.** [Regular Language] *The language $\mathcal{C}$ recognized by a DFA is the set of all words accepted by it: $\mathcal{C} \triangleq \{\boldsymbol{w} \in \Sigma^* \mid \delta^*(q_0, \boldsymbol{w}) \in \mathcal{F}\}$. A language is regular if it is recognized by a DFA.*

**Masked Diffusion Language Models**. MDLMs generate mask-free (or clean) token sequences from partially/fully masked sequences, by repeatedly unmasking the masked positions (Sahoo et al., 2024).

Let $\mathcal{V}$ be a finite vocabulary of clean tokens, $\perp$ be the mask token, and $\mathcal{X} \triangleq \mathcal{V} \cup \{\perp\}$ the augmented token space. $\mathbf{X} \in \mathcal{X}^N$ denotes a random state (an $N$-length token sequence) and $\mathbf{x}$ is its realization. The masked positions of $\mathbf{x}$ are given by: $\mathcal{M}(\mathbf{x}) \triangleq \{i \in [N] : \mathbf{x}(i) = \perp\}$, and $\mathbf{x}(i)$ is the $i$-th token in $\mathbf{x}$. Starting from a masked state $\mathbf{X}_0 \in \mathcal{X}^N$, an MDLM evolves as a Markov chain $\mathbf{X}_0 \to \mathbf{X}_1 \to \cdots \to \mathbf{X}_t \to \cdots \to \mathbf{X}_T$ over a finite number of denoising steps $T$ to produce a clean state $\mathbf{X}_T \in \mathcal{V}^N$.

The transition (Markov) kernels can be constructed from the three main components of MDLMs. For $\mathbf{x}_t$ with $\mathcal{M}(\mathbf{x}_t) \neq \varnothing$, the *denoiser* outputs categoricals $\mathrm{Cat}_i(\cdot \mid \mathbf{x}_t) \in \Delta^{|\mathcal{V}|-1}$ over $i \in \mathcal{M}(\mathbf{x}_t)$, where $\Delta^{|\mathcal{V}|-1}$ is the $(|\mathcal{V}|-1)$-simplex. The token predictions are conditionally independent given $\mathbf{x}_t$, so the denoiser's posterior factorizes over $\mathcal{M}(\mathbf{x}_t)$. The *scheduler* retains some of these factors by drawing a *reveal set* $\mathbf{R}_{t+1} = r$ from some policy $s_{t+1}(\cdot \mid \mathbf{x}_t)$ such that $r \subseteq \mathcal{M}(\mathbf{x}_t)$ and is non-empty. The set

[‡] $\delta^*$ is the DFA's transition function $\delta$ extended from symbols in $\Sigma$ to words in $\Sigma^*$ (Farzan et al., 2022).

$r$ is the subset of $\mathbf{x}_t$'s masked positions that will be revealed for step $t{+}1$. The policy $s_{t+1}(\cdot \mid \mathbf{x}_t)$ can be uniform (Nie et al., 2025), confidence-based (Wu et al., 2026), entropy-based (Ben-Hamu et al., 2025), etc. Lastly, the *decoder* independently samples from $\mathrm{Cat}_i(\cdot \mid \mathbf{x}_t)$ for $i \in r$, and commits the sampled tokens (retaining $\mathbf{x}_t$'s tokens in the remaining positions) to produce the next state $\mathbf{x}_{t+1}$. Since denoising is monotone unmasking, we also use $\mathcal{R}(\mathbf{x}_{t+1}, \mathbf{x}_t) \triangleq \mathcal{M}(\mathbf{x}_t) \setminus \mathcal{M}(\mathbf{x}_{t+1})$ to denote the revealed set of positions between steps $t$ and $t{+}1$.

**Definition 2.2.** [Native Joint Kernel] *Transition from $\mathbf{X}_t = \mathbf{x}_t$ to $\mathbf{X}_{t+1} = \mathbf{x}_{t+1}$ and $\mathbf{R}_{t+1} = r$ is:*

$$\kappa_{t+1}(\mathbf{x}_{t+1}, r \mid \mathbf{x}_t) \triangleq \underbrace{s_{t+1}(r \mid \mathbf{x}_t)}_{\text{drawn reveal set}} \times \underbrace{[\![\mathbf{x}_{t+1} \equiv_{\neg r} \mathbf{x}_t]\!]}_{\text{MDLMs denoise by unmasking}} \times \underbrace{\prod_{i \in r} \mathrm{Cat}_i(\mathbf{x}_{t+1}(i) \mid \mathbf{x}_t)}_{\text{denoiser predictions over masks}}. \quad (1)$$

MDLMs can denoise only by unmasking which necessitates $[\![\mathbf{x}_{t+1} \equiv_{\neg r} \mathbf{x}_t]\!]$ in Definition 2.2[§]. The native kernel $\kappa_{t+1}(\mathbf{x}_{t+1} \mid \mathbf{x}_t)$ is obtained by marginalizing out $\mathbf{R}_{t+1}$ from Eq. (1). A denoising trajectory $\mathbf{x}_{1:T}$ is then a sequence of states $(\mathbf{x}_1, \ldots, \mathbf{x}_T)$ derived through repeated applications of the native kernels $\kappa_{t+1}$ on the corresponding intermediate states $\mathbf{x}_t$ starting from the initial state $\mathbf{x}_0$.

**Definition 2.3.** [Native Path Law] *The distribution of denoising trajectories conditioned on $\mathbf{x}_0$:*

$$p^{\mathsf{mdlm}}_{1:T}(\mathbf{x}_{1:T} \mid \mathbf{x}_0) \triangleq \mathbb{P}(\mathbf{X}_1 = \mathbf{x}_1, \ldots, \mathbf{X}_T = \mathbf{x}_T \mid \mathbf{X}_0 = \mathbf{x}_0) = \prod_{t<T} \kappa_{t+1}(\mathbf{x}_{t+1} \mid \mathbf{x}_t). \quad (2)$$

**Twisted Sequential Monte Carlo**. Sequential Monte Carlo (SMC) is used for approximating trajectory distributions as it performs importance corrections incrementally while building the trajectory (Liu, 2004b)[¶]. It approximates them using weighted particles, by propagating, reweighting, and resampling them as trajectories grow. It implements the Feynman–Kac model (Del Moral, 2004).

**Definition 2.4.** [Feynman–Kac Model] *A Feynman–Kac model defines an unnormalized trajectory distribution $\widetilde{\pi}_{0:t}$ for $t \le T$, as a product of proposal kernels $M$ and non-negative potentials $G$:*

$$\widetilde{\pi}_{0:t}(\mathbf{x}_{0:t}) \triangleq M_0(\mathbf{x}_0)\, G_0(\mathbf{x}_0) \times \prod_{s<t} M_{s+1}(\mathbf{x}_{s+1} \mid \mathbf{x}_s)\, G_{s+1}(\mathbf{x}_s, \mathbf{x}_{s+1}). \quad (3)$$

However, when the target includes only a terminal weight, standard SMC receives no guidance during intermediate steps. Twisted SMC uses twist functions to move this information into incremental potentials, enabling particles to be guided as their trajectories grow (Whiteley & Lee, 2014).

**Definition 2.5.** [Twist-Constructed Potentials] *Consider a Markov path distribution $P(\mathbf{x}_{0:t}) = \nu_0(\mathbf{x}_0) \prod_{s<t} \kappa_{s+1}(\mathbf{x}_{s+1} \mid \mathbf{x}_s)$ and a non-negative terminal weight $h$. Let $\eta_{0:T}$ be twist functions such that $\eta_t > 0$ for $t < T$ and $\eta_T = h$. Given proposal kernels $M$, the potentials are defined as:*

$$G_0(\mathbf{x}_0) \triangleq \frac{\nu_0(\mathbf{x}_0)}{M_0(\mathbf{x}_0)} \eta_0(\mathbf{x}_0), \qquad G_{t+1}(\mathbf{x}_t, \mathbf{x}_{t+1}) \triangleq \frac{\kappa_{t+1}(\mathbf{x}_{t+1} \mid \mathbf{x}_t)}{M_{t+1}(\mathbf{x}_{t+1} \mid \mathbf{x}_t)} \frac{\eta_{t+1}(\mathbf{x}_{t+1})}{\eta_t(\mathbf{x}_t)}. \quad (4)$$

Substituting the potentials from Eq. (4) into Eq. (3) gives $\widetilde{\pi}_{0:t}(\mathbf{x}_{0:t}) = P(\mathbf{x}_{0:t})\, \eta_t(\mathbf{x}_t)$. Therefore, the model targets $P(\mathbf{x}_{0:T}) h(\mathbf{x}_T)$ at $T$. The intermediate twists do not alter it, but move its information into the incremental potentials used to guide particles. The ideal twist at $\mathbf{x}_t$ is the expected terminal weight under the remaining native transitions, i.e., the exact lookahead or continuation probabilities.

## 3 Step-Exact Does Not Mean Trajectory-Exact

Constrained decoding in MDLMs operates along two axes: state positions $i \in [N]$ and denoising steps $0 \le t < T$. At denoising step $t$, a step-exact decoder draws a clean sequence $\mathbf{Y} \in \mathcal{V}^N$ from the denoiser's automaton-constrained posterior (conditioned on $\mathbf{X}_t$) that satisfies the regular language constraint $\mathcal{C}$, i.e., $\mathbf{Y} \in \mathcal{C}$. Only $\mathbf{Y}$'s positions in the reveal set $\mathbf{R}_{t+1}$ are committed to $\mathbf{X}_{t+1}$, while the remaining are discarded. A draw can therefore be exact at every step without the resulting trajectory $\mathbf{X}_{0:T}$ following the native path law (Definition 2.3) conditioned on constraint-satisfaction $\mathbf{X}_T \in \mathcal{C}$.

Throughout the paper, we assume: (1) complete reveal $\mathcal{M}(\mathbf{x}_T) = \varnothing$ at the terminal denoising step $T$; and (2) full support of the denoiser's predictions $\mathrm{Cat}_i(v \mid \mathbf{x}) > 0$ for all $v \in \mathcal{V}$ and for all $i \in \mathcal{M}(\mathbf{x})$. We follow Koo et al. (2024) to lift the DFA from symbols in $\Sigma$ to tokens in the model's vocabulary $\mathcal{V}$.

[§] We use $[N]$ as the universe for set operations of a state's positions: $\neg r \triangleq [N] \setminus r$, $\neg \mathcal{M}(\mathbf{x}) \triangleq [N] \setminus \mathcal{M}(\mathbf{x})$, etc.

[¶] Importance Sampling performs importance corrections only after a trajectory is built (Liu, 2004a).

We now formally characterize the trajectory bias introduced by step-exact decoding. We start by defining the *target path law*, i.e., the native decoder's path law conditioned on constraint-satisfaction, using the Doob $h$-transform (Section 3.1). Next, we derive the transition kernels of the step-exact decoder (Section 3.2), and show that the involved partition sums are lookaheads (valid-continuation probabilities) under a frozen denoiser, and thus step-exact decoding is a frozen Doob transform re-frozen at every step (Section 3.3). Finally, we show that re-freezing is the trajectory bias (Section 3.4).

**Definition 3.1.** [Unbiased Constrained Decoder] *For an initial state $\mathbf{x}_0$ and constraint $\mathcal{C}$, let $\Omega_{\mathcal{C}}(\mathbf{x}_0)$ be the support of native trajectories $\mathbf{x}_{1:T}$ satisfying $\mathbf{x}_T \in \mathcal{C}$. A constrained decoder with path law $q_{1:T}(\cdot \mid \mathbf{x}_0; \mathcal{C})$ is unbiased if its support is $\Omega_{\mathcal{C}}(\mathbf{x}_0)$ and, for every $\mathbf{x}'_{1:T}, \mathbf{x}''_{1:T} \in \Omega_{\mathcal{C}}(\mathbf{x}_0)$,*

$$\frac{q_{1:T}(\mathbf{x}'_{1:T} \mid \mathbf{x}_0; \mathcal{C})}{q_{1:T}(\mathbf{x}''_{1:T} \mid \mathbf{x}_0; \mathcal{C})} = \frac{p^{\mathsf{mdlm}}_{1:T}(\mathbf{x}'_{1:T} \mid \mathbf{x}_0)}{p^{\mathsf{mdlm}}_{1:T}(\mathbf{x}''_{1:T} \mid \mathbf{x}_0)},$$

*i.e., the constrained decoder does not alter the relative probabilities of constraint-valid trajectories.*

### 3.1 Doob-Transformed Decoder

In order to characterize trajectory bias, we first need the path law of a constrained decoder that is conditioned to satisfy $\mathcal{C}$, and is unbiased in the sense of Definition 3.1. The native path law $p^{\mathsf{mdlm}}_{1:T}(\cdot \mid \mathbf{X}_0)$ is unconstrained and so does not satisfy $\mathcal{C}$ by itself. We obtain the unbiased constrained decoder by terminally conditioning $p^{\mathsf{mdlm}}_{1:T}(\mathbf{X}_{1:T} \mid \mathbf{X}_0)$ on satisfying $\mathcal{C}$, i.e., $\mathbf{X}_T \in \mathcal{C}$; using the Doob $h$-transform (Doob, 1957). The Doob $h$-transform conditions Markov processes to satisfy specific constraints, by tilting their transition kernels towards states compatible with highly-probable constraint-valid continuations that are given by the lookaheads $h_t(\mathbf{x}) \triangleq \mathbb{P}(\mathbf{X}_T \in \mathcal{C} \mid \mathbf{X}_t = \mathbf{x})$.

**Definition 3.2.** [Doob Kernel] *For the native kernels $\kappa_{t+1}$ (Definition 2.2) and their exact lookahead $h_t(\mathbf{x})$, with $h_t(\mathbf{x}) > 0$ for all $\mathbf{x}$ in the support of $\kappa_{t+1}$, their Doob $h$-transform gives:*

$$\kappa^{\star}_{t+1}(\mathbf{x}_{t+1} \mid \mathbf{x}_t; \mathcal{C}) \triangleq \kappa_{t+1}(\mathbf{x}_{t+1} \mid \mathbf{x}_t) \times \frac{h_{t+1}(\mathbf{x}_{t+1})}{h_t(\mathbf{x}_t)}. \tag{5}$$

*$h_t(\mathbf{x}) \triangleq \sum_{\mathbf{x}' \in \mathcal{X}^N} \kappa_{t+1}(\mathbf{x}' \mid \mathbf{x})\, h_{t+1}(\mathbf{x}')$, is the exact lookahead of $\kappa_{t+1}$ and $h_T(\mathbf{x}) \triangleq [\![\mathbf{x} \in \mathcal{C}]\!]$.*

**Proposition 3.3.** [Doob Path Law] *The Doob path law is the product of the Doob kernels:*

$$p^{\star}_{1:T}(\mathbf{x}_{1:T} \mid \mathbf{x}_0; \mathcal{C}) \;=\; p^{\mathsf{mdlm}}_{1:T}(\mathbf{x}_{1:T} \mid \mathbf{x}_0) \times \frac{[\![\mathbf{x}_T \in \mathcal{C}]\!]}{h_0(\mathbf{x}_0)}. \tag{6}$$

*Proof sketch.* The product of the Doob kernels $\kappa^{\star}_{t+1}$ for $t < T$, gives the native path law from Definition 2.3; and the $h$-ratios telescope to $[\![\mathbf{x}_T \in \mathcal{C}]\!]/h_0(\mathbf{x}_0)$, thus giving Eq. (6). □

Importantly, the Doob path law $p^{\star}_{1:T}(\mathbf{X}_{1:T} \mid \mathbf{x}_0; \mathcal{C})$ from Proposition 3.3 is unbiased as it prunes all trajectories with terminal state $\mathbf{X}_T \notin \mathcal{C}$, but does so without altering the relative probabilities of the valid trajectories under $p^{\mathsf{mdlm}}_{1:T}(\mathbf{X}_{1:T} \mid \mathbf{x}_0)$. This follows from Eq. (6), where the Doob path law is proportional to $p^{\mathsf{mdlm}}_{1:T}(\mathbf{X}_{1:T} \mid \mathbf{x}_0)$ up to a factor $h_0(\mathbf{x}_0)$ which is constant for a fixed initial state $\mathbf{x}_0$. The Doob terminal law can be obtained by marginalizing out the intermediate states $\mathbf{X}_1, \ldots, \mathbf{X}_{T-1}$ of $p^{\star}_{1:T}(\mathbf{X}_{1:T} \mid \mathbf{x}_0; \mathcal{C})$. Let $Z^{\star}(\mathbf{x}_0) \triangleq h_0(\mathbf{x}_0)$, then:

$$p^{\star}_T(\mathbf{y} \mid \mathbf{x}_0; \mathcal{C}) \;=\; p^{\mathsf{mdlm}}_T(\mathbf{X}_T = \mathbf{y} \mid \mathbf{x}_0) \times \frac{[\![\mathbf{y} \in \mathcal{C}]\!]}{Z^{\star}(\mathbf{x}_0)}, \tag{7}$$

gives the probability that the terminal MDLM output is a clean sequence $\mathbf{y}$, conditioned on the initial state $\mathbf{x}_0$ and on the terminal output satisfying the regular language constraint. When $Z^{\star}(\mathbf{x}_0)$ is not too small, this distribution can be sampled efficiently using rejection sampling from the native decoder.

### 3.2 Automaton-Constrained Step-Exact Decoder

Unlike the Doob-transformed decoder which conditions the native path law on the final state $\mathbf{X}_T$ satisfying the constraint $\mathcal{C}$, the step-exact decoder constrains the denoiser's mean-field posterior at each denoising step to satisfy $\mathcal{C}$ and samples it exactly using FFBS (Dang & Ermon, 2026).

Since the denoiser's published categoricals are conditionally independent given a state $\mathbf{x}$, its mean-field posterior $\mu(\cdot \mid \mathbf{x})$ factorizes over all the positions of $\mathbf{x}$ (respecting already committed positions)

as:

$$\mu(\mathbf{Y}=\mathbf{y} \mid \mathbf{X}=\mathbf{x}) \;=\; [\![\mathbf{y} \equiv_{\neg\mathcal{M}(\mathbf{x})} \mathbf{x}]\!] \times \prod\nolimits_{i\in\mathcal{M}(\mathbf{x})} \mathrm{Cat}_i(\mathbf{y}(i) \mid \mathbf{x}). \tag{8}$$

Sampling the mean-field posterior conditioned on a state $\mathbf{x}$ gives a clean sequence $\mathbf{y} \in \mathcal{V}^N$. This is because we retain $\mathbf{x}$'s tokens for non-masked positions while sampling from the mean-field's factors for masked positions. However, this clean sequence may not be in the regular language $\mathcal{C}$, i.e., $\mathbf{y} \notin \mathcal{C}$. Since the next intermediate state is derived from $\mathbf{x}$ and $\mathbf{y}$, it may not satisfy the regular constraint either. Therefore, the step-exact decoder constrains the mean-field posterior to satisfy $\mathcal{C}$ at each step.

**Proposition 3.4.** [Automaton-Constrained Posterior] *Let the DFA $\mathcal{A}=\langle\Sigma, \mathcal{Q}, q_0, \mathcal{F}, \delta\rangle$ recognize $\mathcal{C}$, and $\mathbf{Q}\in\mathcal{Q}^{N+1}$ be a random state chain of $\mathcal{A}$, then the automaton-constrained posterior is:*

$$\mu(\mathbf{Y}=\mathbf{y}, \mathbf{Q}=\mathbf{q} \mid \mathbf{X}=\mathbf{x}; \mathcal{C}) \;\propto\; \mu(\mathbf{y} \mid \mathbf{x}) \times \psi_{\mathcal{A}}(\mathbf{y}, \mathbf{q}), \tag{9}$$

*where $\psi_{\mathcal{A}}(\mathbf{y}, \mathbf{q})$ is the constraint imposed by $\mathcal{A}$ and is given by:*

$$\psi_{\mathcal{A}}(\mathbf{y}, \mathbf{q}) \;=\; [\![\mathbf{q}(0)=q_0]\!] \times [\![\mathbf{q}(N)\in\mathcal{F}]\!] \times \prod\nolimits_{i\in[N]} [\![\mathbf{q}(i) = \delta(\mathbf{q}(i-1), \mathbf{y}(i))]\!]. \tag{10}$$

The automaton-constrained posterior in Proposition 3.4 is a Conditional Random Field (CRF) (Lafferty et al., 2001) with denoiser categoricals as unary potentials and the DFA $\mathcal{A}$'s transition function providing the compatibility factor for the transfer or pairwise potentials. Since $\mathcal{A}$ is deterministic, every constraint-satisfying $\mathbf{y}\in\mathcal{C}$ corresponds to exactly one DFA state chain for which $\psi_{\mathcal{A}}(\mathbf{y}, \mathbf{q})=1$. Thus, marginalizing out $\mathbf{Q}$ gives $\sum_{\mathbf{q}\in\mathcal{Q}^{N+1}} \psi_{\mathcal{A}}(\mathbf{y}, \mathbf{q}) = [\![\mathbf{y}\in\mathcal{C}]\!]$.

**Definition 3.5.** [Clamped Partition Sum] *For a state $\mathbf{x}'$ reachable from $\mathbf{x}$, i.e., $[\![\mathbf{x}' \equiv_{\neg\mathcal{R}(\mathbf{x}', \mathbf{x})} \mathbf{x}]\!]$, the constraint-valid mass of the completions of $\mathbf{x}'$ under the denoiser conditioned on $\mathbf{x}$ is:*

$$Z(\mathbf{x}'; \mathbf{x}) \;\triangleq\; \sum\nolimits_{\mathbf{y}\in\mathcal{V}^N} [\![\mathbf{y}\in\mathcal{C}]\!] \times [\![\mathbf{y} \equiv_{\neg\mathcal{M}(\mathbf{x}')} \mathbf{x}']\!] \times \prod\nolimits_{i\in\mathcal{M}(\mathbf{x}')} \mathrm{Cat}_i(\mathbf{y}(i) \mid \mathbf{x}). \tag{11}$$

*When $\mathbf{x}' = \mathbf{x}$, the clamped partition sum $Z(\mathbf{x}; \mathbf{x})$ is called the local partition sum.*

The local partition sum $Z(\mathbf{x}; \mathbf{x})$ is the normalizer of the constrained CRF in Proposition 3.4, and scores the constraint-valid probability mass of the masked positions of $\mathbf{x}$ with the denoiser conditioned on the same state $\mathbf{x}$, while the clamped partition sum $Z(\mathbf{x}'; \mathbf{x})$ scores it for the masked positions of $\mathbf{x}'$, but with the denoiser conditioned on a different state $\mathbf{x}$. Following Dang & Ermon (2026), we use FFBS both to sample exactly from the constrained CRF and to compute the partition sums.

**Proposition 3.6.** [Step-Exact Kernel] *For the native kernels $\kappa_{t+1}$ and partition sums $Z(\mathbf{x}; \mathbf{x})$ with $Z(\mathbf{x}; \mathbf{x}) > 0$ for all $\mathbf{x}$ in the support of $\kappa_{t+1}$, the step-exact transition kernel is given by:*

$$\kappa^{\mathcal{A}}_{t+1}(\mathbf{x}_{t+1} \mid \mathbf{x}_t; \mathcal{C}) \;= \kappa_{t+1}(\mathbf{x}_{t+1} \mid \mathbf{x}_t) \times \frac{Z(\mathbf{x}_{t+1}; \mathbf{x}_t)}{Z(\mathbf{x}_t; \mathbf{x}_t)}. \tag{12}$$

*Proof sketch.* Substituting the constrained CRF into the native joint kernel factors out $\kappa_{t+1}$ and leaves $Z(\mathbf{x}_{t+1}; \mathbf{x}_t)$ as the valid-completion sum. Formal proof in Appendix A. □

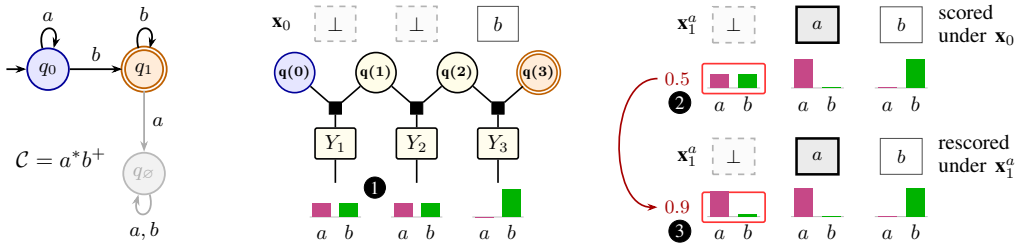
Interestingly, the step-exact kernel in Eq. (12) looks structurally similar to the Doob kernel in Eq. (5), in the sense that both tilt the native kernel $\kappa_{t+1}$ by a ratio of successor to current constraint-valid probability masses, with the partition sums $Z$ of step-exact kernels paralleling the exact lookaheads $h$ of the Doob kernels. We next show that this structural similarity is not a coincidence, but a consequence of the partition sums $Z$ being lookaheads, but of a different Markov chain.

### 3.3 Partition Sums are Lookaheads of a Frozen Denoiser Markov Chain

The exact lookahead $h_t$ is intractable because the native Markov chain reconditions the denoiser on new states for each step. In contrast, a clamped partition sum $Z(\mathbf{x}_{t+1}; \mathbf{x}_t)$ scores the valid completions of $\mathbf{x}_{t+1}$ by conditioning the denoiser only once, at $\mathbf{x}_t$. We characterize their similarity by *freezing* the denoiser at a reference state $\mathbf{x}^*$. Then the frozen native joint kernel for $\mathbf{x}_t$ reachable from $\mathbf{x}^*$ is:

$$\kappa^{(\mathbf{x}^*)}_{t+1}(\mathbf{x}_{t+1}, r_{t+1} \mid \mathbf{x}_t) \;\triangleq\; s_{t+1}(r_{t+1} \mid \mathbf{x}_t) \times [\![\mathbf{x}_{t+1} \equiv_{\neg r_{t+1}} \mathbf{x}_t]\!] \times \prod\nolimits_{i\in r_{t+1}} \mathrm{Cat}_i(\mathbf{x}_{t+1}(i) \mid \mathbf{x}^*).$$

Here, the denoiser's conditioning is fixed at $\mathbf{x}^*$ instead of the current state $\mathbf{x}_t$ as in Definition 2.2. The scheduler is still evaluated at $\mathbf{x}_t$. The frozen terminal law is independent of the order of unmasking since the predicted categoricals are fixed due to freezing the denoiser, yielding a tractable lookahead.




**(a)** DFA for $\mathcal{C}$. **(b)** Automaton-constrained posterior. **(c)** Successor $\mathbf{x}_1^a$ rescoring at $t=0$ and 1.

**Figure 2:** A three-token example of trajectory bias. Let $\mathcal{V}=\{a,b\}$, $\mathcal{C}=a^*b^+$, $T=2$, and $\mathbf{x}_0=(\perp,\perp,b)$, with position 2 revealed before position 1. Revealing $a$ gives $\mathbf{x}_1^a=(\perp,a,b)$, with $aab$ as its only valid completion under $\mathcal{C}$, while revealing $b$ gives $\mathbf{x}_1^b=(\perp,b,b)$, whose completions are both valid. Assuming uniform categoricals at $\mathbf{x}_0$ (❶), $Z(\mathbf{x}_1^a;\mathbf{x}_0)=0.5$ (❷) and $Z(\mathbf{x}_1^b;\mathbf{x}_0)=1$, so the step-exact decoder chooses $\mathbf{x}_1^a$ with probability $0.5/(0.5+1)=1/3$. After committing $\mathbf{x}_1^a$, the denoiser rescores position 1 as $\mathrm{Cat}_1(a\mid\mathbf{x}_1^a)=0.9$ (❸), so $\mathbf{x}_1^a$'s actual valid mass is $Z(\mathbf{x}_1^a;\mathbf{x}_1^a)=0.9$; which is what the Doob path law uses to choose $\mathbf{x}_1^a$, $h_1(\mathbf{x}_1^a)=0.9$, i.e., with probability $0.9/(0.9+1)=9/19$. But step-exact decoding only renormalizes by $Z(\mathbf{x}_1^a;\mathbf{x}_1^a)$ in the next step, thus never correcting $Z(\mathbf{x}_1^a;\mathbf{x}_0)$. Since $aab$ is the only valid completion of $\mathbf{x}_1^a$, $p^{\mathcal{A}}_{1:2}((\mathbf{x}_1^a,aab)\mid\mathbf{x}_0;\mathcal{C})=1/3$ but $p^{\star}_{1:2}((\mathbf{x}_1^a,aab)\mid\mathbf{x}_0;\mathcal{C})=9/19$, due to the per-step mismatch $Z(\mathbf{x}_1^a;\mathbf{x}_0)/Z(\mathbf{x}_1^a;\mathbf{x}_1^a)=5/9$ in Eq. (14).

**Proposition 3.7.** [Frozen Lookahead] *The lookahead for the constraint-satisfying terminal law induced by the frozen native kernels, for any state $\mathbf{x}_t$ reachable from $\mathbf{x}^*$ is:*

$$\widehat{h}_t^{(\mathbf{x}^*)}(\mathbf{x}_t)=\sum\nolimits_{\mathbf{x}\in\mathcal{X}^N}[\![\mathbf{x}\equiv_{\neg\mathcal{M}(\mathbf{x}_t)}\mathbf{x}_t]\!]\times[\![\mathbf{x}\in\mathcal{C}]\!]\times\prod\nolimits_{i\in\mathcal{M}(\mathbf{x}_t)}\mathrm{Cat}_i(\mathbf{x}(i)\mid\mathbf{x}^*). \tag{13}$$

*Therefore, $\widehat{h}_t^{(\mathbf{x}_t)}(\mathbf{x}_t)=Z(\mathbf{x}_t;\mathbf{x}_t)$ and $\widehat{h}_{t+1}^{(\mathbf{x}_t)}(\mathbf{x}_{t+1})=Z(\mathbf{x}_{t+1};\mathbf{x}_t)$.*

*Proof sketch.* Monotone unmasking and posterior sampling are independent of the scheduler, giving Eq. (13). The two identities follow from Eq. (11) for $\mathbf{x}^*=\mathbf{x}_t$. Formal proof in Appendix B. □

From Proposition 3.7, both partition sums in Eq. (12) are thus lookaheads of the Markov chain frozen at the current state, which agrees with the native chain on the first transition, i.e., when $\mathbf{x}^*=\mathbf{x}_t$.

**Proposition 3.8.** [Re-Frozen Doob Transforms] *Each step-exact kernel is the exact Doob transform of the frozen native kernels at its current state $\mathbf{x}_t$:*

$$\kappa^{\mathcal{A}}_{t+1}(\mathbf{x}_{t+1}\mid\mathbf{x}_t;\mathcal{C})\;=\;\kappa_{t+1}(\mathbf{x}_{t+1}\mid\mathbf{x}_t)\times\frac{\widehat{h}_{t+1}^{(\mathbf{x}_t)}(\mathbf{x}_{t+1})}{\widehat{h}_t^{(\mathbf{x}_t)}(\mathbf{x}_t)}.$$

Proposition 3.8 applies to the chain frozen at the current state. At the next step, however, the denoiser is re-frozen at the new state, replacing the expected successor $\widehat{h}_{t+1}^{(\mathbf{x}_t)}(\mathbf{x}_{t+1})$ with $\widehat{h}_{t+1}^{(\mathbf{x}_{t+1})}(\mathbf{x}_{t+1})$. It is this re-freezing that prevents the composition of the step-exact kernels from being the Doob transform of a single Markov chain. Since the partition sums are computed at each step, an intermediate state $\mathbf{x}_{t+1}$ is thus scored twice, by $Z(\mathbf{x}_{t+1};\mathbf{x}_t)$ as a successor (e.g., ❷ in Figure 2c) and by $Z(\mathbf{x}_{t+1};\mathbf{x}_{t+1})$ as the current state (e.g., ❸ in Figure 2c) at step $t+1$. We next show how this causes trajectory bias.

### 3.4 Trajectory Bias

We identify the tilt of the path law induced by composing the step-exact kernels, relative to the Doob path law, by multiplying them along a trajectory, yielding the native path law and the tilt factor:

$$\prod\nolimits_{t<T}\frac{Z(\mathbf{x}_{t+1};\mathbf{x}_t)}{Z(\mathbf{x}_t;\mathbf{x}_t)}\;=\;\left[\frac{Z(\mathbf{x}_1;\mathbf{x}_0)}{Z(\mathbf{x}_0;\mathbf{x}_0)}\times\frac{Z(\mathbf{x}_2;\mathbf{x}_1)}{Z(\mathbf{x}_1;\mathbf{x}_1)}\times\frac{Z(\mathbf{x}_3;\mathbf{x}_2)}{Z(\mathbf{x}_2;\mathbf{x}_2)}\times\cdots\times\frac{Z(\mathbf{x}_T;\mathbf{x}_{T-1})}{Z(\mathbf{x}_{T-1};\mathbf{x}_{T-1})}\right]$$

The tilt factor is a product of ratios of partition sums. Since all positions are revealed at $T$, Eq. (11) gives $Z(\mathbf{x}_T;\mathbf{x}_{T-1})=[\![\mathbf{x}_T\in\mathcal{C}]\!]$, the Doob terminal weight of Eq. (6). Each highlighted pair holds the two scores of an intermediate state $\mathbf{x}_t$, scored as a successor $Z(\mathbf{x}_t;\mathbf{x}_{t-1})$ and as the current state $Z(\mathbf{x}_t;\mathbf{x}_t)$. We call the ratio $Z(\mathbf{x}_t;\mathbf{x}_t)/Z(\mathbf{x}_t;\mathbf{x}_{t-1})$ the *re-freezing ratio* of $\mathbf{x}_t$. It measures how re-freezing changes the valid mass estimate of $\mathbf{x}_t$'s completions. If it is greater than 1, then the valid mass of $\mathbf{x}_t$ was underestimated, so paths through $\mathbf{x}_t$ are under-weighted (and vice versa if less than 1):

**Proposition 3.9.** [Re-Freezing Tilt] *The step-exact decoder samples the Doob path law tilted by the inverse re-freezing ratios of its intermediate states (for trajectories in common support):*

$$p^{\mathcal{A}}_{1:T}(\mathbf{x}_{1:T} \mid \mathbf{x}_0; \mathcal{C}) = p^{\star}_{1:T}(\mathbf{x}_{1:T} \mid \mathbf{x}_0; \mathcal{C}) \times \frac{Z^{\star}(\mathbf{x}_0)}{Z(\mathbf{x}_0; \mathbf{x}_0)} \times \prod_{t=1}^{T-1} \frac{Z(\mathbf{x}_t; \mathbf{x}_{t-1})}{Z(\mathbf{x}_t; \mathbf{x}_t)}. \tag{14}$$

*Proof sketch.* Substitute the step-exact and Doob kernels into the ratio of their path laws. □

Since $Z^{\star}(\mathbf{x}_0)/Z(\mathbf{x}_0; \mathbf{x}_0)$ is constant, the two path laws align if every re-freezing ratio equals 1, i.e.,

$$Z(\mathbf{x}_t; \mathbf{x}_{t-1}) = Z(\mathbf{x}_t; \mathbf{x}_t), \qquad \text{for every } \; 0 < t < T.$$

However, this generally does not hold, since unmasking tokens changes the denoiser's predictions for $\mathcal{M}(\mathbf{x}_t)$, and this is structural to the working of MDLMs itself. Trajectory bias thus manifests due to scoring the same positions under two differently conditioned denoisers. Corollary 3.10 gives the exact condition, which only requires the product of ratios to be constant.

**Corollary 3.10.** [Trajectory Exactness] *The step-exact decoder is unbiased (Definition 3.1) if and only if the product of re-freezing ratios is constant over $\Omega_{\mathcal{C}}(\mathbf{x}_0)$, i.e., for every $\mathbf{x}_{1:T} \in \Omega_{\mathcal{C}}(\mathbf{x}_0)$:*

$$\prod_{t=1}^{T-1} \frac{Z(\mathbf{x}_t; \mathbf{x}_t)}{Z(\mathbf{x}_t; \mathbf{x}_{t-1})} \;=\; \frac{Z^{\star}(\mathbf{x}_0)}{Z(\mathbf{x}_0; \mathbf{x}_0)}. \tag{15}$$

**Corollary 3.11.** [No Bias for Vacuous Constraints] *For a vacuous regular constraint $\mathcal{C} = \mathcal{V}^N$ (does not impose any restrictions), Eq. (16) holds for every $t < T$, and there is no trajectory bias.*

*Proof.* Formal proof is in Appendix C.2. □

**Corollary 3.12.** [No Bias for One-Step Reveal All] *When all masked positions in every initial state $\mathbf{x}_0$ are revealed in a single step, i.e., $T = 1$, Eq. (16) holds and there is no trajectory bias.*

*Proof.* Formal proof is in Appendix C.3. □

The trajectory bias introduced by the tilt is correctable because: (1) the tilt factorizes over the steps as shown in Eq. (14) allowing it to be corrected incrementally at each step; and (2) a re-freezing ratio can be computed efficiently and exactly at its step $t-1$ when $\mathbf{x}_t$ is reached by computing the clamped partition sum $Z(\mathbf{x}_t; \mathbf{x}_{t-1})$ from the FFBS messages that sampled $\mathbf{x}_t$, and the next step's local partition sum $Z(\mathbf{x}_t; \mathbf{x}_t)$ from the denoiser query at $\mathbf{x}_t$ that is needed at the next step $t$ anyway.

## 4 TWISTER: AUTOMATON-TWISTED SEQUENTIAL MONTE CARLO

Since the re-freezing tilt factorizes over steps (Proposition 3.9), it can be corrected incrementally at each step. We first show mathematically that the tilt can be corrected via our proposed Feynman–Kac corrector, and then we describe our TWISTER algorithm implementing it.

### 4.1 FEYNMAN–KAC CORRECTOR FOR THE RE-FREEZING TILT

We use the tractable step-exact kernels from Proposition 3.8 as the proposals: $M_{t+1}(\mathbf{x}_{t+1} \mid \mathbf{x}_t) \triangleq \kappa^{\mathcal{A}}_{t+1}(\mathbf{x}_{t+1} \mid \mathbf{x}_t; \mathcal{C})$, and correct for the re-freezing ratio formalized in Section 3.4 with an exact twist.

**Definition 4.1.** [Feynman–Kac Twist] *The Feynman–Kac twist is given by:*

$$\eta_t(\mathbf{x}) \;\triangleq\; \begin{cases} Z(\mathbf{x}; \mathbf{x}), & t < T, \\ [\![\mathbf{x} \in \mathcal{C}]\!], & t = T \end{cases}$$

The ideal twist is the exact lookahead $h_t(\mathbf{x})$ from Definition 3.2, which requires future denoiser calls and is thus intractable. The local partition sum $Z(\mathbf{x}; \mathbf{x})$ is the frozen lookahead $\widehat{h}^{(\mathbf{x})}_t(\mathbf{x})$ from Proposition 3.7, i.e., a tractable surrogate of $h_t(\mathbf{x})$ from a single denoiser call at $\mathbf{x}$ at step $t$. From Definition 4.1, the Feynman–Kac twists are just the local partition sums at each step, thus they are exactly computable using the same FFBS already used for sampling the proposals $M_{t+1}$. The potentials $G_{t+1}$ of the Feynman–Kac model can then be constructed from the twists by Eq. (4) as:

$$G_0(\mathbf{x}_0) = Z(\mathbf{x}_0; \mathbf{x}_0), \qquad G_{t+1}(\mathbf{x}_t, \mathbf{x}_{t+1}) = \frac{\kappa_{t+1}(\mathbf{x}_{t+1} \mid \mathbf{x}_t)}{M_{t+1}(\mathbf{x}_{t+1} \mid \mathbf{x}_t)} \times \frac{\eta_{t+1}(\mathbf{x}_{t+1})}{\eta_t(\mathbf{x}_t)} \;=\; \frac{Z(\mathbf{x}_{t+1}; \mathbf{x}_{t+1})}{Z(\mathbf{x}_{t+1}; \mathbf{x}_t)},$$

**Algorithm 1** TWISTER for step-exact decoding (adaptive resampling shown in Appendix E).

**Require:** DFA $\mathcal{A}$ recognizing $\mathcal{C}$; initial state $\mathbf{x}_0$; number of denoising steps $T$; number of SMC particles $K$

Query denoiser at $\mathbf{x}_0$;
$(\mathbf{x}_0^k, W_0^k) \leftarrow (\mathbf{x}_0, 1/K)$ for all $k \in [K]$
**for** $t = 0, \ldots, T-1$ **do**
  **STEP-EXACT FFBS PROPAGATE**
  **parallel for** $k = 1, \ldots, K$ **do**
    Sample $\mathbf{x}_{t+1}^k \sim M_{t+1}(\cdot \mid \mathbf{x}_t^k)$ ▷ *use cached FFBS messages*
    $\widehat{Z}_{t+1}^k \leftarrow Z(\mathbf{x}_{t+1}^k; \mathbf{x}_t^k)$ ▷ *predecessor clamped-partition sum*
    **if** $t < T-1$ **then**
      Query denoiser at $\mathbf{x}_{t+1}^k$; $Z_{t+1}^k \leftarrow Z(\mathbf{x}_{t+1}^k; \mathbf{x}_{t+1}^k)$ ▷ *successor partition, cache FFBS messages*
    **else**
      $Z_T^k \leftarrow [\![\mathbf{x}_T^k \in \mathcal{C}]\!]$
  **RE-FREEZING CORRECTION**
  **parallel for** $k = 1, \ldots, K$ **do**
    $G_{t+1}^k \leftarrow Z_{t+1}^k / \widehat{Z}_{t+1}^k$
    $\widetilde{W}_{t+1}^k \leftarrow W_t^k G_{t+1}^k$
  $W_{t+1}^k \leftarrow \widetilde{W}_{t+1}^k / \sum_{k'=1}^K \widetilde{W}_{t+1}^{k'}$ for all $k \in [K]$
**return** $(\mathbf{x}_T^{1:K}, W_T^{1:K})$

i.e., the re-freezing ratio of $\mathbf{x}_{t+1}$, for $t < (T-1)$, and $G_T = 1$ since $Z(\mathbf{x}_T; \mathbf{x}_{T-1}) = [\![\mathbf{x}_T \in \mathcal{C}]\!]$.

**Theorem 4.2.** [Twist Fixes the Tilt] *The Feynman–Kac twists correct the re-freezing tilt (Proposition 3.9), thus making the Feynman–Kac model target the Doob path law:*

$$G_0(\mathbf{x}_0) \prod_{t<T} M_{t+1}(\mathbf{x}_{t+1} \mid \mathbf{x}_t)\; G_{t+1}(\mathbf{x}_t, \mathbf{x}_{t+1}) \;=\; p^{\star}_{1:T}(\mathbf{x}_{1:T} \mid \mathbf{x}_0; \mathcal{C})\; Z^{\star}(\mathbf{x}_0)$$

*Proof sketch.* Substitute for $M_{t+1}$ and $G_{t+1}$ in Definition 2.4. Formal proof in Appendix D. □

In Figure 2, weighting $\mathbf{x}_1^a$ by its re-freezing ratio $9/5$ turns the step-exact choice $1/3 : 2/3$ into the Doob choice $9/19 : 10/19$.

### 4.2 TWISTER ALGORITHM

Algorithm 1 captures TWISTER's working. It requires a DFA $\mathcal{A}$ (lifted to the model's vocabulary $\mathcal{V}$) that recognizes the regular language $\mathcal{C}$; an initial state $\mathbf{x}_0$, i.e., a token block containing the prompt and masked tokens; the number of denoising steps $T$; and the number of SMC particles $K$.

TWISTER starts by querying the denoiser at $\mathbf{x}_0$, and caching its FFBS messages. Following this, it initializes all $K$ SMC particles to $\mathbf{x}_0$ with weight $1/K$. The cached FFBS messages are used to sample from the Feynman–Kac proposal $M_{t+1}$ (Proposition 3.8), and compute the predecessor clamped-partition sum $\widehat{Z}_{t+1}^k$. For each propagated nonterminal particle, the denoiser is queried at the successor state to compute the successor partition sum $Z_{t+1}^k$ and cache the FFBS messages for the next step. The ratio $G_{t+1}^k = Z_{t+1}^k / \widehat{Z}_{t+1}^k$ gives the exact Feynman–Kac potential required for fixing the re-freezing tilt (Section 4.1). Each incoming particle weight is multiplied by this potential, and the resulting weights are normalized. Particles are jointly resampled with their caches and assigned uniform weight when resampling is triggered (described in Appendix E).

### 4.3 TRAJECTORY BIAS EXPERIMENT

We measure trajectory bias using regular language constraints defined over token classes. The number of token classes is an experiment configured parameter. Next, we rank the tokens by the probabilities assigned to them by the model conditioned on a fully masked input. We select the most probable tokens and then partition them equally across the token classes. Tokens outside this pool are assigned to the token class $x$. A model's generation can then be projected onto a small alphabet of symbols, i.e., token classes $\{a, b, c, d, x\}$, etc..

We use such token classes to build regular expressions which are then used to constrain the model's generation. We permute the token classes to ensure that the results do not depend on a particular token-to-class mapping. Trajectory bias is then measured by using either Dang & Ermon (2026), TWISTER$_{\texttt{smc4}}$, or TWISTER$_{\texttt{smc8}}$ to sample from the automaton-constrained distribution subject to the regular expression; relative to the native decoder conditioned on constraint satisfaction approximated using rejection sampling (Eq. (7)). We use pairwise total variation distance to measure the similarity between the two distributions. We also compare independent splits of the rejection samples to estimate the finite-sample noise floor. Results are averaged with equal weight across partition configurations. Thus, deviations above the rejection noise floor measure trajectory bias rather than token-level validity or Monte Carlo error. For each model and $T \in \{1, 2, 4, 8, 16\}$ denoising steps, we generate 40,000 native samples and 10,000 constrained samples at temperature 1 under random remasking. The one-step reveal for $T = 1$ is an experimental setting of Corollary 3.12. We evaluate on the models DREAM-7B-BASE, DREAM-7B-INST, DREAMCODER-7B-INST, LLADA-8B-BASE, LLADA-8B-INST, and OWT-130M.

## 5 RELATED WORK

**Constrained Decoding for Autoregressive Models**. Most recent approaches consist of maintaining a parser or automaton over the generated prefix and masking out invalid next tokens at each step (Willard & Louf, 2023; Koo et al., 2024; Beurer-Kellner et al., 2024; Geng et al., 2023; Ugare et al., 2024; Dong et al., 2025). While this guarantees constraint satisfaction by construction, Park et al. (2024) showed that such locally constrained decoding distorts the model's distribution over valid sequences. To address this, Loula et al. (2025) approximate the globally constrained posterior with SMC, combining locally constrained decoding with incremental reweighting and resampling to correct the resulting bias. Dang et al. (2026) further improve this setup by constructing stronger automaton-based proposals that encode future constraint satisfaction, yielding faster SMC convergence with fewer particles. Like these approaches, we use SMC to target the model's globally constrained distribution rather than local constraint enforcement alone, but we focus on the less-explored domain of constrained decoding for MDLMs. Unlike locally constrained LLM decoding, which ignores future valid mass, step-exact MDLM decoding accounts for this mass exactly, but under a denoiser frozen at the current state.

**Constrained Decoding for Diffusion Models**. Suresh et al. (2025) proposed DINGO as the first constrained decoder for MDLMs with formal guarantees. DINGO uses dynamic programming over automaton states to select a constraint-satisfying block that maximizes the per-step mean-field probability. Dang & Ermon (2026) further generalize DINGO's approach by replacing per-step MAP with exact sampling from the automaton-constrained mean-field posterior at each denoising step. Like DINGO, their method guarantees that the constraint is satisfied by construction, but it further supports stochastic sampling under arbitrary remasking schedules. Crucially, both the DINGO and Dang & Ermon (2026) still constrain generation *only* at individual steps. We show that such local exactness does not, however, preserve the model's relative probabilities over valid multi-step denoising trajectories. We address this gap by introducing TWISTER as the first automaton-twisted SMC constrained decoder for MDLMs that targets the trajectory-exact globally constrained path law. Hasan et al. (2025) apply Feynman–Kac SMC correctors to steer discrete diffusion sampling at inference time through temperature scaling or an external reward. Luo et al. (2026) likewise use SMC for MDLMs, but reweight particles by trajectory-level confidence to improve sample quality. In contrast, TWISTER uses SMC to enforce formal constraints.

## 6 CONCLUSION

We showed that sampling locally from an MDLM's automaton-constrained posterior, despite being exact at each denoising step, still tilts the distribution relative to the globally constrained distribution when composed across steps. We derived this trajectory bias as a product of local and clamped-partition sum ratios, which are a consequence of rescoring the constraint-valid completions of the same intermediate states, but under different denoiser conditionings. We showed that the tilt factorizes across the denoising steps, and thus can be corrected incrementally at each step. We then introduced TWISTER, an automaton-twisted SMC constrained decoder that corrects this tilt by using the step-exact decoder as its proposal. Further, for regular-language constraints, its incremental Feynman–Kac potentials are exactly computable from partition sums already obtained during step-exact sampling.

### AI USE STATEMENT

We have not used generative AI tools for this work.

### REPRODUCIBILITY STATEMENT

Our code and data will be made publicly available upon acceptance of the paper.

## REFERENCES


Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas Nolte, and Brian Karrer. Accelerated sampling from masked diffusion models via entropy bounded unmasking. In *NeurIPS*, 2025. doi: 10.48550/arXiv.2505.24857.

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Guiding LLMs the right way: fast, non-invasive constrained generation. In *ICML*, 2024. doi: 10.48550/arXiv.2403.06988.

C. K. Carter and R. Kohn. On Gibbs sampling for state space models. *Biometrika*, 81(3):541–553, 1994. doi: 10.1093/biomet/81.3.541.

Meihua Dang and Stefano Ermon. Constrained decoding for diffusion language models via efficient inference over finite automata. *arXiv preprint arXiv:2607.07026*, 2026. doi: 10.48550/arXiv.2607.07026.

Meihua Dang, Linxin Song, Honghua Zhang, Jieyu Zhao, Guy Van den Broeck, and Stefano Ermon. Mitigating bias in locally constrained decoding via tractable proposals. In *ICML*, 2026. doi: 10.48550/arXiv.2606.01926.

Pierre Del Moral. *Feynman-Kac Formulae*, pp. 47–93. Springer, 2004. ISBN 978-1-4684-9393-1. doi: 10.1007/978-1-4684-9393-1_2.

Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. XGrammar: Flexible and efficient structured generation engine for large language models. In *MLSys*, 2025. doi: 10.48550/arXiv.2411.15100.

Joseph L Doob. Conditional brownian motion and the boundary limits of harmonic functions. *Bulletin de la Société mathématique de France*, 85:431–458, 1957. doi: 10.24033/bsmf.1494.

Azadeh Farzan, Dominik Klumpp, and Andreas Podelski. Sound sequentialization for concurrent program verification. In *PLDI*, pp. 506–521, 2022. doi: 10.1145/3519939.3523727.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured NLP tasks without finetuning. In *EMNLP*, 2023.

Mohsin Hasan, Marta Skreta, Alan Aspuru-Guzik, Yoshua Bengio, and Kirill Neklyudov. Discrete Feynman-Kac correctors. In *AI4Math@ICML*, 2025. doi: 10.48550/arXiv.2601.10403.

Terry Koo, Frederick Liu, and Luheng He. Automata-based constraints for language model decoding. In *COLM*, 2024. doi: 10.48550/arXiv.2407.08103.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pp. 282–289, 2001.

Jun S. Liu. *Basic Principles: Rejection, Weighting, and Others*, pp. 23–52. Springer, 2004a. ISBN 978-0-387-76371-2. doi: 10.1007/978-0-387-76371-2_2.

Jun S. Liu. *Theory of Sequential Monte Carlo*, pp. 53–77. Springer, 2004b. ISBN 978-0-387-76371-2. doi: 10.1007/978-0-387-76371-2_3.

João Loula, Benjamin LeBrun, Li Du, Ben Lipkin, Clemente Pasti, Gabriel Grand, Tianyu Liu, Yahya Emara, Marjorie Freedman, Jason Eisner, Ryan Cotterell, Vikash Mansinghka, Alexander K. Lew, Tim Vieira, and Timothy J. O'Donnell. Syntactic and semantic control of large language models via sequential Monte Carlo. In *ICLR*, 2025. doi: 10.48550/arXiv.2504.13139.

Ziwei Luo, Ziqi Jin, Lei Wang, Lidong Bing, and Thomas B. Schön. Self-rewarding sequential Monte Carlo for masked diffusion language models. *arXiv preprint arXiv:2602.01849*, 2026. doi: 10.48550/arXiv.2602.01849.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. In *NeurIPS*, 2025. doi: 10.48550/arXiv.2502.09992.

NousResearch. json-mode-eval. `https://huggingface.co/datasets/NousResearch/json-mode-eval`, 2024.

Kanghee Park, Jiayu Wang, Taylor Berg-Kirkpatrick, Nadia Polikarpova, and Loris D'Antoni. Grammar-aligned decoding. In *NeurIPS*, 2024. doi: 10.48550/arXiv.2405.21047.

Subham Sekhar Sahoo, Marianne Arriola, Aaron Gokaslan, Edgar Mariano Marroquin, Alexander M Rush, Yair Schiff, Justin T Chiu, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *NeurIPS*, 2024. doi: 10.48550/arXiv.2406.07524.

Tarun Suresh, Debangshu Banerjee, Shubham Ugare, Sasa Misailovic, and Gagandeep Singh. DINGO: Constrained inference for diffusion LLMs. In *NeurIPS*, 2025. doi: 10.48550/arXiv.2505.23061.

Shubham Ugare, Tarun Suresh, Hangoo Kang, Sasa Misailovic, and Gagandeep Singh. SynCode: LLM generation with grammar augmentation. *TMLR*, 2024. doi: 10.48550/arXiv.2403.01632.

Nick Whiteley and Anthony Lee. Twisted particle filters. *The Annals of Statistics*, 42(1):115–141, 2014. doi: 10.1214/13-AOS1167.

Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*, 2023. doi: 10.48550/arXiv.2307.09702.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dLLM: Training-free acceleration of diffusion LLM by enabling KV cache and parallel decoding. In *ICLR*, 2026. doi: 10.48550/arXiv.2505.22618.

Zhihui Xie, Jiacheng Ye, Lin Zheng, Jiahui Gao, Jingwei Dong, Zirui Wu, Xueliang Zhao, Shansan Gong, Xin Jiang, Zhenguo Li, et al. Dream-coder 7B: An open diffusion language model for code. *arXiv preprint arXiv:2509.01142*, 2025. doi: 10.48550/arXiv.2509.01142.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7B: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025. doi: 10.48550/arXiv.2508.15487.

## APPENDIX CONTENTS

**A Proof for Step-Exact Kernel** **13**

**B Proof for Frozen Lookahead** **13**

**C Proofs for Trajectory-Bias Boundary Cases** **14**

C.1 Kernel-Level Trajectory Exactness . . . . . 14

C.2 Vacuous Constraints . . . . . 14

C.3 One-Step Reveal All . . . . . 14

**D Proof for Twist Fixes the Tilt** **15**

**E TWISTER Algorithm (With Adaptive Resampling)** **15**

**F Constraint Satisfaction Experiment** **16**

## A Proof for Step-Exact Kernel

**Proposition 3.6.** [Step-Exact Kernel] *For the native kernels $\kappa_{t+1}$ and partition sums $Z(\mathbf{x};\mathbf{x})$ with $Z(\mathbf{x};\mathbf{x}) > 0$ for all $\mathbf{x}$ in the support of $\kappa_{t+1}$, the step-exact transition kernel is given by:*

$$\kappa^{\mathcal{A}}_{t+1}(\mathbf{x}_{t+1} \mid \mathbf{x}_t; \mathcal{C}) \;= \kappa_{t+1}(\mathbf{x}_{t+1} \mid \mathbf{x}_t) \times \frac{Z(\mathbf{x}_{t+1};\mathbf{x}_t)}{Z(\mathbf{x}_t;\mathbf{x}_t)}.$$

*Proof.* The joint step-exact kernel $\kappa^{\mathcal{A}}_{t+1}(\mathbf{x}', r \mid \mathbf{x}_t; \mathcal{C})$ is given by:

$$\begin{aligned}
&= \; s_{t+1}(r \mid \mathbf{x}_t)\, [\![\mathbf{x}' \equiv_{\neg r} \mathbf{x}_t]\!] \;\times\; \sum\nolimits_{\mathbf{y},\mathbf{q}} \mu(\mathbf{y},\mathbf{q} \mid \mathbf{x}_t; \mathcal{C})\, [\![\mathbf{x}' \equiv_r \mathbf{y}]\!] \\
&= \; \frac{s_{t+1}(r \mid \mathbf{x}_t)\, [\![\mathbf{x}' \equiv_{\neg r} \mathbf{x}_t]\!]}{Z(\mathbf{x}_t;\mathbf{x}_t)} \;\times\; \sum\nolimits_{\mathbf{y},\mathbf{q}} \mu(\mathbf{y} \mid \mathbf{x}_t)\, \psi_{\mathcal{A}}(\mathbf{y},\mathbf{q})\, [\![\mathbf{x}' \equiv_r \mathbf{y}]\!] && \text{by Eq. (9)} \\
&= \; \kappa_{t+1}(\mathbf{x}', r \mid \mathbf{x}_t) \;\times\; \frac{Z(\mathbf{x}';\mathbf{x}_t)}{Z(\mathbf{x}_t;\mathbf{x}_t)} && \text{by Eq. (11)}
\end{aligned}$$

The step-exact kernel is obtained by marginalizing out the reveal set $\mathbf{R}$ from the joint kernels. □

## B Proof for Frozen Lookahead

**Proposition 3.7.** [Frozen Lookahead] *The lookahead for the constraint-satisfying terminal law induced by the frozen native kernels, for any state $\mathbf{x}_t$ reachable from $\mathbf{x}^*$ is:*

$$\widehat{h}^{(\mathbf{x}^*)}_t(\mathbf{x}_t) = \sum\nolimits_{\mathbf{x}\in\mathcal{X}^N} [\![\mathbf{x} \equiv_{\neg\mathcal{M}(\mathbf{x}_t)} \mathbf{x}_t]\!] \times [\![\mathbf{x} \in \mathcal{C}]\!] \times \prod\nolimits_{i\in\mathcal{M}(\mathbf{x}_t)} \mathrm{Cat}_i(\mathbf{x}(i) \mid \mathbf{x}^*).$$

*Therefore, $\widehat{h}^{(\mathbf{x}_t)}_t(\mathbf{x}_t) = Z(\mathbf{x}_t;\mathbf{x}_t)$ and $\widehat{h}^{(\mathbf{x}_t)}_{t+1}(\mathbf{x}_{t+1}) = Z(\mathbf{x}_{t+1};\mathbf{x}_t)$..*

*Proof.* Let $\mathcal{R}_t$ be the set of valid reveal-set sequences from $t$ to $T$. Then $\mathbb{P}_{\mathbf{x}^*}(\mathbf{X}_T = \mathbf{x} \,|\, \mathbf{X}_t = \mathbf{x}_t)$ is

$$\begin{aligned}
&= \sum_{r_{t+1:T}\in\mathcal{R}_t} \prod_{s=t}^{T-1} \kappa^{(\mathbf{x}^*)}_{s+1}(\mathbf{x}_{s+1}, r_{s+1} \mid \mathbf{x}_s) \\
&= [\![\mathbf{x} \equiv_{\neg\mathcal{M}(\mathbf{x}_t)} \mathbf{x}_t]\!] \prod_{i\in\mathcal{M}(\mathbf{x}_t)} \mathrm{Cat}_i(\mathbf{x}(i) \mid \mathbf{x}^*) \times \sum_{r_{t+1:T}\in\mathcal{R}_t} \prod_{s=t}^{T-1} s_{s+1}(r_{s+1} \mid \mathbf{x}_s), \\
&= [\![\mathbf{x} \equiv_{\neg\mathcal{M}(\mathbf{x}_t)} \mathbf{x}_t]\!] \prod_{i\in\mathcal{M}(\mathbf{x}_t)} \mathrm{Cat}_i(\mathbf{x}(i) \mid \mathbf{x}^*), && \text{by} \sum_{r_{t+1:T}\in\mathcal{R}_t} \prod_{m=t}^{T-1} s_{m+1}(r_{m+1} \mid \mathbf{x}_m) = 1
\end{aligned}$$

Therefore, the frozen lookahead is given by:

$$\begin{aligned}
\widehat{h}^{(\mathbf{x}^*)}_t(\mathbf{x}_t) &= \sum_{\mathbf{x}\in\mathcal{X}^N} [\![\mathbf{x} \in \mathcal{C}]\!]\, \mathbb{P}_{\mathbf{x}^*}(\mathbf{X}_T = \mathbf{x} \mid \mathbf{X}_t = \mathbf{x}_t) \\
&= \sum_{\mathbf{x}\in\mathcal{X}^N} [\![\mathbf{x} \equiv_{\neg\mathcal{M}(\mathbf{x}_t)} \mathbf{x}_t]\!][\![\mathbf{x} \in \mathcal{C}]\!] \prod_{i\in\mathcal{M}(\mathbf{x}_t)} \mathrm{Cat}_i(\mathbf{x}(i) \mid \mathbf{x}^*), && \text{giving Eq. (13)}
\end{aligned}$$

Substituting $\mathbf{x}^* = \mathbf{x}_t$ in Eq. (11) gives $\widehat{h}^{(\mathbf{x}_t)}_t(\mathbf{x}_t) = Z(\mathbf{x}_t;\mathbf{x}_t)$ and $\widehat{h}^{(\mathbf{x}_t)}_{t+1}(\mathbf{x}_{t+1}) = Z(\mathbf{x}_{t+1};\mathbf{x}_t)$. □

# C Proofs for Trajectory-Bias Boundary Cases

## C.1 Kernel-Level Trajectory Exactness

**Lemma C.1.** [Kernel-Level Trajectory Exactness] *The step-exact path law composed of step-exact kernels $\kappa^{\mathcal{A}}_{t+1}$ equals that composed of the Doob kernels $\kappa^{\star}_{t+1}$ if and only if, for every $t < T$, every state $\mathbf{x}_t$ reachable with non-zero probability under the path law, with $Z(\mathbf{x}_t;\mathbf{x}_t)>0$ and $h_t(\mathbf{x}_t)>0$, and every $\mathbf{x}_{t+1}$ in the support of $\kappa_{t+1}(\cdot \mid \mathbf{x}_t)$:*

$$\frac{Z(\mathbf{x}_{t+1};\mathbf{x}_t)}{Z(\mathbf{x}_t;\mathbf{x}_t)} \;=\; \frac{h_{t+1}(\mathbf{x}_{t+1})}{h_t(\mathbf{x}_t)} \tag{16}$$

## C.2 Vacuous Constraints

**Corollary 3.11.** [No Bias for Vacuous Constraints] *For a vacuous constraining regular language $\mathcal{C}=\mathcal{V}^N$, i.e., constrains nothing, Eq.* (16) *holds for every $t<T$, and there is no trajectory bias.*

*Proof.* Since $\mathcal{C} = \mathcal{V}^N$, every terminal state $\mathbf{X}_T \in \mathcal{C}$, and so $h_T(\mathbf{x}) = 1$ for all $\mathbf{x} \in \mathcal{V}^N$. From backward recursion suppose $h_{t+1}(\mathbf{x}') = 1$ for every $\mathbf{x}'$.

$$\begin{aligned}
h_t(\mathbf{x}) &= \sum\nolimits_{\mathbf{x}'} \kappa_{t+1}(\mathbf{x}' \mid \mathbf{x})\;\; h_{t+1}(\mathbf{x}') && \text{by Definition 3.2}\\
&= \sum\nolimits_{\mathbf{x}'} \kappa_{t+1}(\mathbf{x}' \mid \mathbf{x}), && \text{by the backward-induction hypothesis}\\
&= 1
\end{aligned}$$

The local partition sum $Z(\mathbf{x};\mathbf{x})$ is the normalizer of Eq. (9) and so can be written as:

$$\begin{aligned}
Z(\mathbf{x};\mathbf{x}) &= \sum\nolimits_{\mathbf{y},\mathbf{q}} \mu(\mathbf{y} \mid \mathbf{x})\;\; \psi_{\mathcal{A}}(\mathbf{y},\mathbf{q})\\
&= \sum\nolimits_{\mathbf{y}} \mu(\mathbf{y} \mid \mathbf{x}), && \text{since } \mathcal{C}=\mathcal{V}^N \text{ and } \mathcal{A} \text{ is deterministic}\\
&= \sum\nolimits_{\mathbf{y}} [\![\mathbf{y} \equiv_{\neg\mathcal{M}(\mathbf{x})} \mathbf{x}]\!] \;\prod\nolimits_{i\in\mathcal{M}(\mathbf{x})} \mathrm{Cat}_i(\mathbf{y}(i) \mid \mathbf{x}), && \text{from Eq. (8)}\\
&= \prod\nolimits_{i\in\mathcal{M}(\mathbf{x})} \sum\nolimits_{v\in\mathcal{V}} \mathrm{Cat}_i(v \mid \mathbf{x}), && \text{by } \textstyle\sum_{v\in\mathcal{V}} \mathrm{Cat}_i(v \mid \mathbf{x}) = 1\\
&= 1
\end{aligned}$$

The clamped partition sum $Z(\mathbf{x}';\mathbf{x})$ is given as:

$$\begin{aligned}
Z(\mathbf{x}';\mathbf{x}) &= \sum\nolimits_{\mathbf{y}} [\![\mathbf{y} \in \mathcal{C}]\!]\; [\![\mathbf{y} \equiv_{\neg\mathcal{M}(\mathbf{x}')} \mathbf{x}']\!] \;\prod\nolimits_{i\in\mathcal{M}(\mathbf{x}')} \mathrm{Cat}_i(\mathbf{y}(i) \mid \mathbf{x}), && \text{from Eq. (11)}\\
&= \sum\nolimits_{\mathbf{y}} [\![\mathbf{y} \equiv_{\neg\mathcal{M}(\mathbf{x}')} \mathbf{x}']\!] \;\prod\nolimits_{i\in\mathcal{M}(\mathbf{x}')} \mathrm{Cat}_i(\mathbf{y}(i) \mid \mathbf{x}), && \text{since } \mathcal{C}=\mathcal{V}^N\\
&= \prod\nolimits_{i\in\mathcal{M}(\mathbf{x}')} \sum\nolimits_{v\in\mathcal{V}} \mathrm{Cat}_i(v \mid \mathbf{x}), && \text{by } \textstyle\sum_{v\in\mathcal{V}} \mathrm{Cat}_i(v \mid \mathbf{x}) = 1\\
&= 1
\end{aligned}$$

Therefore $Z(\mathbf{x}';\mathbf{x}) = h_{t+1}(\mathbf{x}') = Z(\mathbf{x};\mathbf{x}) = h_t(\mathbf{x}) = 1$ for all $t<T$. □

## C.3 One-Step Reveal All

**Corollary 3.12.** [No Bias for One-Step Reveal All] *For a finite horizon $T=1$, where every initial realization $\mathbf{x}_0$ is completely unmasked in one step, Eq.* (16) *holds and there is no trajectory bias.*

*Proof.* Since $T=1$, we have $h_1(\mathbf{x}') = [\![\mathbf{x}' \in \mathcal{C}]\!]$ from Definition 3.2, and so $h_0(\mathbf{x}_0)$ is given by:

$$\begin{aligned}
h_0(\mathbf{x}_0) &= \sum\nolimits_{\mathbf{x}'} \kappa_1(\mathbf{x}' \mid \mathbf{x}_0)\;\; h_1(\mathbf{x}')\\
&= \sum\nolimits_{\mathbf{x}'} \kappa_1(\mathbf{x}' \mid \mathbf{x}_0)\; [\![\mathbf{x}' \in \mathcal{C}]\!], && \text{by } h_1(\mathbf{x}') = [\![\mathbf{x}' \in \mathcal{C}]\!]
\end{aligned}$$

The clamped partition sum $Z(\mathbf{x}';\mathbf{x}_0)$ is given by Eq. (11) as:

$$\begin{aligned} Z(\mathbf{x}';\mathbf{x}_0) &= \sum\nolimits_{\mathbf{y}} [\![\mathbf{y}\in\mathcal{C}]\!]\ [\![\mathbf{y}\equiv_{\neg\mathcal{M}(\mathbf{x}')}\mathbf{x}']\!]\ \prod\nolimits_{i\in\mathcal{M}(\mathbf{x}')}\mathrm{Cat}_i(\mathbf{y}(i)\mid\mathbf{x}_0) \\ &= \sum\nolimits_{\mathbf{y}} [\![\mathbf{y}\in\mathcal{C}]\!]\ [\![\mathbf{x}'=\mathbf{y}]\!], && \text{since } \mathcal{M}(\mathbf{x}')=\varnothing \text{ for } T=1 \\ &= [\![\mathbf{x}'\in\mathcal{C}]\!] \end{aligned}$$

The local partition sum $Z(\mathbf{x}_0;\mathbf{x}_0)$ is the normalizer of Eq. (12) and so can be written as:

$$\begin{aligned} Z(\mathbf{x}_0;\mathbf{x}_0) &= \sum\nolimits_{\mathbf{x}'} \kappa_1(\mathbf{x}'\mid\mathbf{x}_0)\ \ Z(\mathbf{x}';\mathbf{x}_0) \\ &= \sum\nolimits_{\mathbf{x}'} \kappa_1(\mathbf{x}'\mid\mathbf{x}_0)\ [\![\mathbf{x}'\in\mathcal{C}]\!], && \text{by } Z(\mathbf{x}';\mathbf{x}_0)=[\![\mathbf{x}'\in\mathcal{C}]\!] \end{aligned}$$

Since $Z(\mathbf{x}';\mathbf{x}_0)=h_1(\mathbf{x}')$ and $Z(\mathbf{x}_0;\mathbf{x}_0)=h_0(\mathbf{x}_0)$,. □

## D Proof for Twist Fixes the Tilt

**Theorem 4.2.** [Twist Fixes the Tilt] *The Feynman–Kac twists correct the re-freezing tilt (Proposition 3.9), thus making the step-exact decoder path law target the Doob path law:*

$$G_0(\mathbf{x}_0)\prod\nolimits_{t<T} M_{t+1}(\mathbf{x}_{t+1}\mid\mathbf{x}_t)\ G_{t+1}(\mathbf{x}_t,\mathbf{x}_{t+1}) \ = \ p^{\star}_{1:T}(\mathbf{x}_{1:T}\mid\mathbf{x}_0;\mathcal{C})\ Z^{\star}(\mathbf{x}_0).$$

*Proof.* Substituting the twist-constructed potentials from Eq. (4) into the left-hand side gives:

$$\begin{aligned} & G_0(\mathbf{x}_0)\prod\nolimits_{t<T} M_{t+1}(\mathbf{x}_{t+1}\mid\mathbf{x}_t)\ G_{t+1}(\mathbf{x}_t,\mathbf{x}_{t+1}) \\ &= \eta_0(\mathbf{x}_0)\prod\nolimits_{t<T}\kappa_{t+1}(\mathbf{x}_{t+1}\mid\mathbf{x}_t)\ \frac{\eta_{t+1}(\mathbf{x}_{t+1})}{\eta_t(\mathbf{x}_t)} \\ &= \prod\nolimits_{t<T}\kappa_{t+1}(\mathbf{x}_{t+1}\mid\mathbf{x}_t)\ \eta_T(\mathbf{x}_T) && \text{by telescoping the twists} \\ &= p^{\mathsf{mdlm}}_{1:T}(\mathbf{x}_{1:T}\mid\mathbf{x}_0)\ [\![\mathbf{x}_T\in\mathcal{C}]\!] && \text{by Definition 4.1} \\ &= p^{\star}_{1:T}(\mathbf{x}_{1:T}\mid\mathbf{x}_0;\mathcal{C})\ Z^{\star}(\mathbf{x}_0) && \text{by Eq. (6)} \end{aligned}$$

□

## E Twister Algorithm (With Adaptive Resampling)

Algorithm 2 gives the complete TWISTER algorithm. It requires a DFA $\mathcal{A}$ (lifted to the model's vocabulary $\mathcal{V}$) that recognizes the regular language $\mathcal{C}$; an initial state $\mathbf{x}_0$, i.e., a token block containing the prompt and masked tokens to be unmasked; the maximum number of denoising steps $T$; the number of SMC particles $K$; and the minimum effective sample size $\mathrm{ESS}_{\min}$, i.e., the threshold for triggering resampling of the SMC particles. Resampling is used in SMC to mitigate *weight degeneracy*: when weights of most SMC particles become too small, rendering them useless and thus wasting compute.

TWISTER starts by querying the denoiser at the initial state $\mathbf{x}_0$, computing its partition sum $Z_0$ and caching its FFBS messages. Following this, it initializes all $K$ SMC particles to the initial state $\mathbf{x}_0$ with weight $1/K$. The cached FFBS messages are used to sample from the Feynman–Kac proposal $M_{t+1}$, i.e., the step-exact proposal of Proposition 3.8, and compute the predecessor clamped-partition sum $\widehat{Z}^k_{t+1}$. For each propagated nonterminal particle, the denoiser is queried at the successor state to compute the successor partition sum $Z^k_{t+1}$ and cache the FFBS messages for the next step. The ratio $G^k_{t+1}=Z^k_{t+1}/\widehat{Z}^k_{t+1}$ gives the exact Feynman–Kac potential required in Section 4.1 for fixing the tilt from Proposition 3.9. Each incoming particle weight is multiplied by this potential, and the resulting weights are normalized. Particles are jointly resampled with their caches and assigned uniform weight when resampling is triggered by the effective sample size falling below $\mathrm{ESS}_{\min}K$.

**Complexity**. One weighted-automaton pass costs $\mathcal{O}(N|\mathcal{Q}||\mathcal{V}|)$, and so, the total automaton cost for $K$ SMC particles and $T$ denoising steps is $\mathcal{O}(KTN|\mathcal{Q}||\mathcal{V}|)$, in addition to the denoiser evaluations. TWISTER is parallelizable over the step-exact FFBS propagate and re-freezing correction phases, since the particles are conditionally independent until weight normalization and resampling.

**Algorithm 2** TWISTER for step-exact decoding (with adaptive resampling).

**Require:** DFA $\mathcal{A}$ recognizing $\mathcal{C}$; initial state $\mathbf{x}_0$; number of denoising steps $T$
**Require:** number of SMC particles $K$; minimum effective sample size $\mathrm{ESS}_{\min} \in [0, 1]$

Query the denoiser at $\mathbf{x}_0$
$(\mathbf{x}_0^k, W_0^k) \leftarrow (\mathbf{x}_0, 1/K)$ for all $k \in [K]$
**for** $t = 0, \ldots, T-1$ **do**
  **STEP-EXACT FFBS PROPAGATE**
  **parallel for** $k = 1, \ldots, K$ **do**
    Sample $\mathbf{x}_{t+1}^k \sim M_{t+1}(\cdot \mid \mathbf{x}_t^k)$ ▷ *use cached FFBS messages*
    $\widehat{Z}_{t+1}^k \leftarrow Z(\mathbf{x}_{t+1}^k; \mathbf{x}_t^k)$ ▷ *predecessor clamped-partition sum*
    **if** $t < T-1$ **then**
      Query the denoiser at $\mathbf{x}_{t+1}^k$
      $Z_{t+1}^k \leftarrow Z(\mathbf{x}_{t+1}^k; \mathbf{x}_{t+1}^k)$ ▷ *successor partition sum and cache FFBS messages*
    **else**
      $Z_T^k \leftarrow [\![\mathbf{x}_T^k \in \mathcal{C}]\!]$
  **RE-FREEZING CORRECTION**
  **parallel for** $k = 1, \ldots, K$ **do**
    $G_{t+1}^k \leftarrow Z_{t+1}^k / \widehat{Z}_{t+1}^k$
    $\widetilde{W}_{t+1}^k \leftarrow W_t^k \, G_{t+1}^k$
  $W_{t+1}^k \leftarrow \widetilde{W}_{t+1}^k / \sum_{k'=1}^K \widetilde{W}_{t+1}^{k'}$ for all $k \in [K]$
  **ADAPTIVE RESAMPLING**
  $\mathrm{ESS} \leftarrow \left(\sum_{k'=1}^K (W_{t+1}^{k'})^2\right)^{-1}$
  **if** $t < T-1$ **and** $\mathrm{ESS} < \mathrm{ESS}_{\min} K$ **then**
    $\mathbf{x}_{t+1}^{1:K} \leftarrow \textsc{Resample}(\mathbf{x}_{t+1}^{1:K}, W_{t+1}^{1:K})$ ▷ *resample cache jointly*
    $W_{t+1}^k \leftarrow 1/K$ for all $k \in [K]$
**return** $(\mathbf{x}_T^{1:K}, W_T^{1:K})$

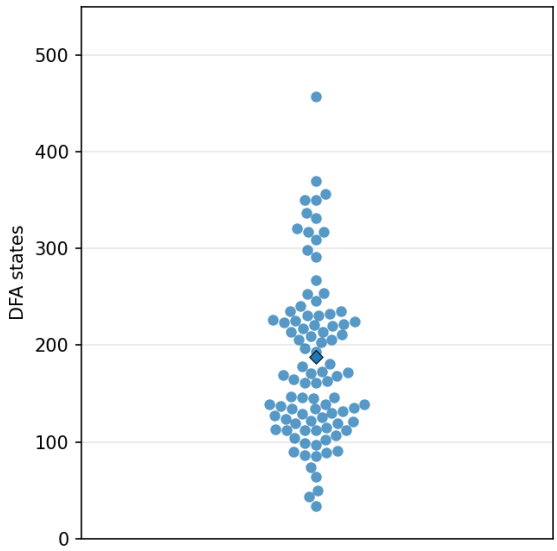



**Figure 3:** Swarm plot of token-prefix DFA state counts on JSON-Mode-Eval. Each point is one datapoint; the diamond marks the mean.

## F CONSTRAINT SATISFACTION EXPERIMENT

```
[system]
You are a helpful assistant that answers in JSON. Here's the json schema you must adhere to:
<schema>
{'title': 'WirelessAccessPoint', 'type': 'object', 'properties': {'ssid': {'title': 'SSID', 'type':
      'string'}, 'securityProtocol': {'title': 'SecurityProtocol', 'type': 'string'}, 'bandwidth': {'title':
      'Bandwidth', 'type': 'string'}}, 'required': ['ssid', 'securityProtocol', 'bandwidth']}
</schema>

[user]
I'm currently configuring a wireless access point for our office network and I need to generate a JSON
      object that accurately represents its settings. The access point's SSID should be 'OfficeNetSecure', it
      uses WPA2-Enterprise as its security protocol, and it's capable of a bandwidth of up to 1300 Mbps on
      the 5 GHz band. This JSON object will be used to document our network configurations and to automate
      the setup process for additional access points in the future. Please provide a JSON object that
      includes these details.
Only output the JSON object, no other text or comments.
```

**Figure 4:** Example JSON-Mode-Eval prompt messages (datapoint `json-0`). The final user-turn line is our appended instruction to emit only the JSON object.

As a sanity check that TWISTER still produces only constraint-satisfying outputs, we evaluate it along with other constrained decoding methods on the JSON-Mode-Eval dataset released by NousResearch (2024). JSON-Mode-Eval consists of 100 zero-shot datapoints, each specifying a given schema and requiring a JSON output that conforms to that schema. Of the 100 datapoints in JSON-Mode-Eval, we exclude six whose schemas cannot be compiled into a regular expression by Outlines (five use a named-root wrapper rather than a valid JSON Schema root, and one is malformed), and evaluate on the remaining 94. For each remaining datapoint, we compile the JSON schema into a regular expression with the Outlines library from Willard & Louf (2023) and transform that regular expression into a token-prefix automaton over the model vocabulary. Figure 3 shows the distribution of token-prefix DFA sizes created for the 94 evaluated schemas. The distribution is concentrated in the low hundreds of states, with most values falling between roughly $100$ and $250$. The mean number of states is 188, with a minimum of 34 states and a maximum of 625 states.

We build the dataset prompt for each datapoint, append an instruction to emit only the JSON object, and then decode under each of the constrained strategies described below. Figure 4 shows an example of the prompt message for a JSON-Mode-Eval datapoint given to the model.

We compare four constrained decoding strategies. DINGO†, at each denoising step, selects a constraint-satisfying completion that maximizes the automaton-weighted mean-field posterior. DINGO† picks the mode by using dynamic programming over automaton states to select a locally optimal constraint-satisfying block under the mean-field factorization. Our DINGO† implementation follows the original dynamic-programming procedure from Suresh et al. (2025). The original code is not publicly available. Dang & Ermon† instead samples exactly from the automaton-constrained mean-field posterior at each step using FFBS (Carter & Kohn, 1994; Dang & Ermon, 2026), which guarantees local validity but is still subject to the trajectory bias characterized in §3.4. Our Dang & Ermon† implementation follows the setup and procedure described in Dang & Ermon (2026). As with DINGO†, the original code is not publicly available. Finally, $\text{TWISTER}_{\text{smc1}}$ and $\text{TWISTER}_{\text{smc4}}$ apply TWISTER with $K=1$ and $K=4$ particles, using the step-exact constrained posterior and correcting trajectory bias with the Feynman–Kac twists described in §4.

We evaluate all decoding methods on six open MDLMs from the DREAM (Ye et al., 2025), DREAMCODER (Xie et al., 2025), and LLaDA (Nie et al., 2025) families: DREAM-7B-BASE, DREAM-7B-INST, DREAMCODER-7B-BASE, DREAMCODER-7B-INST, LLADA-8B-BASE, and LLADA-8B-INST. We chose these models because they are among the strongest publicly available MDLMs. We set generation length and denoising steps to $1.5\times$ the gold answer length per datapoint.

Table 1 shows the results of our evaluation of the different decoding methods on the JSON-Mode-Eval dataset. We report the parse validity, schema validity, and average time in seconds of the generated outputs. Parse validity ensures that the generated output is valid JSON, while schema validity ensures that the generated output conforms to the expected schema set by the specific dataset sample. Importantly, all decoding methods, including TWISTER, consistently produce outputs that are valid JSON and conform to the expected schema. Although we report average generation times, our implementation is not optimized for performance. There are techniques to speed up the decoding time

**Table 1:** Constraint satisfaction on JSON-Mode-Eval for all decoding methods. The *Parse Valid (%)* and *Schema Valid (%)* columns measure whether outputs are valid JSON and satisfy the schema set by the datapoint; *Time (s)* measures the average time taken to generate the output.

| Model | Method | Parse Valid (%) | Schema Valid (%) | Time (s) |
|---|---|---|---|---|
| DREAM-7B-BASE | DINGO† | 100 | 98 | $22 \pm 43$ |
| | Dang & Ermon† | 100 | 98 | $22 \pm 43$ |
| | TWISTER$_{\text{smc1}}$ | 100 | 98 | $32 \pm 65$ |
| | TWISTER$_{\text{smc4}}$ | 100 | 98 | $42 \pm 86$ |
| DREAM-7B-INST | DINGO† | 100 | 99 | $21 \pm 45$ |
| | Dang & Ermon† | 100 | 99 | $21 \pm 46$ |
| | TWISTER$_{\text{smc1}}$ | 100 | 99 | $33 \pm 72$ |
| | TWISTER$_{\text{smc4}}$ | 100 | 99 | $42 \pm 87$ |
| DREAMCODER-7B-BASE | DINGO† | 100 | 98 | $21 \pm 38$ |
| | Dang & Ermon† | 100 | 98 | $21 \pm 38$ |
| | TWISTER$_{\text{smc1}}$ | 100 | 98 | $31 \pm 57$ |
| | TWISTER$_{\text{smc4}}$ | 100 | 98 | $40 \pm 73$ |
| DREAMCODER-7B-INST | DINGO† | 100 | 99 | $22 \pm 47$ |
| | Dang & Ermon† | 100 | 99 | $22 \pm 47$ |
| | TWISTER$_{\text{smc1}}$ | 100 | 99 | $34 \pm 76$ |
| | TWISTER$_{\text{smc4}}$ | 100 | 98 | $42 \pm 88$ |
| LLADA-8B-BASE | DINGO† | 100 | 98 | $19 \pm 33$ |
| | Dang & Ermon† | 100 | 98 | $19 \pm 33$ |
| | TWISTER$_{\text{smc1}}$ | 100 | 98 | $27 \pm 50$ |
| | TWISTER$_{\text{smc4}}$ | 100 | 98 | $36 \pm 61$ |
| LLADA-8B-INST | DINGO† | 100 | 99 | $18 \pm 37$ |
| | Dang & Ermon† | 100 | 99 | $21 \pm 42$ |
| | TWISTER$_{\text{smc1}}$ | 100 | 99 | $29 \pm 62$ |
| | TWISTER$_{\text{smc4}}$ | 100 | 99 | $37 \pm 73$ |

such as those described in Dang & Ermon (2026). This experiment acts as a confirmation to show that the trajectory-level correction we achieve with TWISTER does not compromise the constraint satisfaction already guaranteed by step-exact decoding.