Constrained Decoding for Diffusion Language Models via Efficient Inference over Finite Automata

Meihua Dang
Stanford University
mhdang@cs.stanford.edu

Stefano Ermon
Stanford University
ermon@cs.stanford.edu

Abstract

Constrained decoding is essential for serving LLMs, ensuring that generated outputs follow specific structures such as JSON schema-formatted function calls. Existing systems are designed for autoregressive models and assume left-to-right generation, masking out invalid next tokens at each step. Diffusion language models, however, break this assumption: they sample multiple positions simultaneously from a fully-factorized mean-field distribution at each denoising step. In this paper, we present an exact and tractable algorithm for sampling from the constrained mean-field posterior under constraints expressible as finite automata. Viewing finite automata as graphical models, we obtain tractable representations of the constrained distribution that enable efficient inference. The approach guarantees constraint satisfaction by construction, supports both greedy and sampling-based decoding, and is compatible with parallel and block-wise decoding under arbitrary remasking schedules. Applying depth-reduction techniques from arithmetic circuit theory, we further reduce sampling depth from linear to logarithmic in the sequence length. Empirical evaluations on Dream-7B and LLaDA-8B show substantial accuracy gains across various tasks including function calling (xLAM, BFCL), planning (Sudoku, Countdown), text-to-SQL (Spider), and math reasoning (GSM-Symbolic), with little inference overhead relative to unconstrained decoding. For example, on BFCL-Live, our approach improves Dream-7B's greedy decoding accuracy from 63.9% to 71.5%, and stochastic sampling accuracy from 22.3% to 69.0%, where the unconstrained baseline collapses, with under 5% wall-clock overhead.

1 Introduction

A wide range of LLM applications require outputs to strictly adhere to structural constraints. Examples include generating function calls conforming to a given JSON schema [18, 19] and producing syntactically correct SQL queries [10]. Autoregressive LLMs enforce such constraints at inference time via *constrained decoding*, where invalid next tokens violating the constraint are masked out at each generation step [34]. This approach is now a standard component of LLM inference frameworks such as XGrammar [5], Guidance [7] and SGLang [40].

Recent advances in discrete diffusion language models (dLLMs) have established them as competitive alternatives to autoregressive LLMs [2, 12, 24]. Open-weight models such as LLaDA-8B [15] and Dream-7B [36] have demonstrated performance comparable to similarly sized autoregressive models on code and math reasoning. Closed-source models including Mercury [9], Gemini Diffusion [6], and Seed Diffusion [29] further demonstrate excellent inference efficiency.

Existing constrained generation frameworks [5, 7, 40] are specifically designed for left-to-right autoregressive generation and adapting them to dLLMs poses unique challenges. Unlike autoregressive LLMs, dLLMs sample multiple tokens simultaneously from a fully-factorized mean-field distribution at each step. Because constraints operate globally, masking tokens independently at each position is not enough: locally valid tokens can still combine into invalid sequences.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

To bridge this gap, we introduce an exact and tractable inference algorithm that, at each diffusion step, samples from the *constrained mean-field posterior*: $p_\theta(x^0 \mid x^t, \mathcal{C}) \propto p_\theta(x^0 \mid x^t) \cdot \mathbb{1}[x^0 \in \mathcal{C}]$ where the constraint $\mathcal{C}$ is expressed as a finite automaton, $p_\theta(x^0 \mid x^t)$ is the model's fully factorized mean-field prediction, and the indicator enforces the constraint globally on the complete sequence x^0. Viewing the automaton as a graphical model encoding constraint $\mathcal{C}$ as its support, we are able to obtain a tractable representation for the constrained mean-field prediction $p_\theta(x^0 \mid x^t, \mathcal{C})$, which enables efficient sampling. Our approach guarantees constraint satisfaction by construction, supports both greedy decoding and stochastic sampling, and applies to parallel and block-wise decoding under arbitrary remasking schedules. It also computes per-token marginals under the constrained distribution, which we use as a confidence signal for remasking.

Naive sampling from the chain-structured graphical model induced by a finite automaton requires $\mathcal{O}(L)$ *sequential* forward/backward steps for a sequence of length L. Inspired by depth-reduction techniques for arithmetic circuits [32], we restructure this linear computation graph into a binary tree of prefix–suffix state pairs, reducing inference steps from $\mathcal{O}(L)$ to $\mathcal{O}(\log L)$ while preserving the same sampling distribution. This yields substantially more parallelism on modern GPUs.

Empirically, we evaluate our method on LLaDA-8B and Dream-7B comparing against the unconstrained base models. We consider diverse constraints and tasks, including JSON-schema constraints for function calling (xLAM, BFCL), format constraints for planning (Sudoku, Countdown), SQL syntax and schema constraints (Spider), and symbolic expression constraints for math reasoning (GSM-Symbolic). Our method guarantees constraint satisfaction and substantially improves accuracy. For example, on BFCL-Live, our approach improves Dream's greedy decoding accuracy from 63.9% to 71.5%, and remains robust under stochastic sampling: while the unconstrained Dream collapses to 22.3%, our method achieves 69.0%, with under 5% wall-clock overhead.

2 Related Work

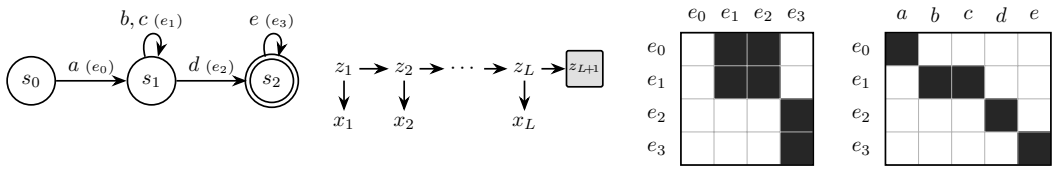
Constrained decoding for autoregressive LLMs. There is an extensive line of work on locally constrained decoding [28, 25, 22, 31, 17], with a primary focus on enforcing hard constraints efficiently during autoregressive generation. These approaches typically operate by masking out invalid next tokens based on the current prefix, thereby ensuring local constraint satisfaction at each decoding step. Existing frameworks leverage different constraint representations and develop different parsing algorithms correspondingly: for example, Outlines [34] uses regular expressions and their corresponding finite automata representations, while more recent systems such as XGrammar [5] and Guidance [7] rely on context-free grammars.

Constrained decoding for dLLMs. DINGO [30] is one pioneering work exploring inference algorithms over finite automata to support constrained decoding for dLLMs. In particular, DINGO leverages a dynamic programming algorithm to perform MAP inference on deterministic finite automata with their inputs weighted by the mean-field predictions from dLLMs, supporting block-wise greedy decoding from dLLMs. Our approach can be viewed as a generalization of DINGO, supporting greedy decoding and sampling under arbitrary masking strategies constrained by (possibly nondeterministic) finite automata (Section 4). Mündler et al. [14] and Zhang et al. [39] target richer constraints expressed as context-free grammars (CFGs). Both adopt a per-step rejection-sampling scheme that draws candidate tokens and then verifies whether the intermediate output remains acceptable; Zhang et al. [39] additionally introduce a lookahead step that uses the mean-field distributions to reliably verify proposed tokens. These methods do not guarantee constraint satisfaction *within a finite sampling budget*, as acceptance rate can be extremely low in practice. CDD [3] takes a complementary approach, enforcing constraints through differentiable optimization during sampling, which requires differentiable relaxations. In contrast, our method enables exact inference under finite automata constraints without requiring such relaxations.

3 Background

3.1 Diffusion Language Models

Masked dLLMs [2, 24, 27] learn to reverse a forward corruption process q that stochastically converts a clean token sequence $x^0 = (x_1^0, \ldots, x_L^0)$ of length L into a noisy sequence x^t over time $t \in [0, 1]$,



(a) DFA $\mathcal{M}$ (b) Graphical Model (c) Transition (d) Emission

Figure 1: **(a)** A finite automaton $\mathcal{M}$ accepting $a(b|c)^*de^*$. **(b)** The induced chain-structured graphical model over length-L sequences, with edge-valued latents $z_{1:L}$ emitting observed tokens $x_{1:L}$. **(c)** The transition factor $p(z_t \mid z_{t-1})$, nonzero when consecutive edges chain end-to-end ($\mathrm{dst}(z_{t-1}) = \mathrm{src}(z_t)$). **(d)** The emission factor $p(x_t \mid z_t)$, nonzero when x_t is in the label set of edge z_t.

by masking each token with probability t:

$$q_{t|0}(x_i^t \mid x_i^0) = \begin{cases} t, & x_i^t = \texttt{[MASK]} \\ 1-t, & x_i^t = x_i^0 \end{cases} \qquad q_{t|0}(x^t \mid x^0) = \prod_i q_{t|0}(x_i^t \mid x_i^0). \tag{1}$$

dLLMs parameterize the reverse process via a mean-field distribution $p_\theta(x^0 \mid x^t)$ over clean tokens given noisy input, trained by maximizing an evidence lower bound on the data log-likelihood [24, 16].

Sampling from dLLMs begins with a fully masked sequence and proceeds by iteratively unmasking tokens. A denoising step from time t to $s < t$ first draws a full clean sequence $\hat{x}^0$ from the mean-field prediction $p_\theta(x^0 \mid x^t)$, then commits a subset $\mathcal{U} \subseteq \{i : x_i^t = \texttt{[MASK]}\}$ of newly predicted positions and re-masks the rest to obtain x^s:

$$\hat{x}^0 \sim p_\theta(x^0 \mid x^t) = \prod_{i=1}^{L} p_\theta(x_i^0 \mid x^t), \qquad x_i^s = \begin{cases} \hat{x}_i^0 & i \in \mathcal{U} \\ x_i^t & \text{otherwise.} \end{cases} \tag{2}$$

Iterating this denoising process from x^T down to x^0 defines a marginal distribution $p_{\mathrm{dLLM}}(x^0)$ over clean sequences, which we refer to as the dLLM's *generative distribution*. While dLLMs are trained on uniformly random masks, in practice decoding heuristics are used at inference time to choose which positions to unmask. For example, Dream selects positions with the lowest predictive entropy at each step [36], while LLaDA selects positions with the highest predictive probability [15].

3.2 Finite Automata

In this work, we consider constraints expressible as *finite automata* [8].

Definition 3.1. A (nondeterministic) finite automaton (FA) is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{V}, \delta, s_0, \mathcal{F})$ with states $\mathcal{S}$, alphabet $\mathcal{V}$ (often corresponding to a dLLM vocabulary), transition function $\delta : \mathcal{S} \times \mathcal{V} \to 2^{\mathcal{S}}$, initial state s_0, and accept states $\mathcal{F} \subseteq \mathcal{S}$. A string $v_{1:L}$ is *accepted* if there exists a sequence of states $s_0, s_1, \ldots, s_L$ with $s_i \in \delta(s_{i-1}, v_i)$ and $s_L \in \mathcal{F}$. A finite automaton is *deterministic* if $|\delta(s, v)| = 1$ for all s and v.

A finite automaton encodes a constraint $\mathcal{C} \subseteq \mathcal{V}^L$ as the set of length-L accepted strings. Equivalently, as illustrated in Figure 1, we can also view $\mathcal{M}$ as a directed graph with *edge set*

$$\mathcal{E} := \{(s, s') \in \mathcal{S} \times \mathcal{S} \mid \exists v \text{ s.t. } s' \in \delta(s, v)\}.$$

For each edge $e = (s, s')$, we write $\mathrm{src}(e) := s$, $\mathrm{dst}(e) := s'$ and $\mathrm{label}(e) := \{v \mid s' \in \delta(s, v)\}$. Under this view, a string $v_{1:L}$ is accepted iff there exists a path $e_{1:L}$ such that $\mathrm{src}(e_1) = s_0$, $\mathrm{dst}(e_i) = \mathrm{src}(e_{i+1})$ for all $i < L$, $\mathrm{dst}(e_L) \in \mathcal{F}$, and $v_i \in \mathrm{label}(e_i)$ for all i.

Finite automata as graphical models. Following Dang et al. [4], we define $p_{\mathcal{M}}$ as a chain-structured graphical model over sequences of L tokens $p_{\mathcal{M}}(x_{1:L})$ such that $p_{\mathcal{M}}(x_{1:L}) > 0$ if and only if $x_{1:L} \in \mathcal{C}$, that is, the constraint $\mathcal{C}$ is encoded as its support. $p_{\mathcal{M}}$ can also be viewed as a hidden Markov model [23], with latent variables $z_i \in \mathcal{E}$ taking values from the edge set of the automaton. See Figure 1 for a visualization.

The latent sequence $z_{1:L}$ encodes a random path through the automaton: z_t is the edge taken at step i with token x_i emitted from its label $\mathrm{label}(e)$. Consecutive edges must chain end-to-end, i.e.

$\mathrm{dst}(z_i) = \mathrm{src}(z_{i+1})$, as required by the transition matrix, starting at the initial state s_0 and ending in an accepting state in $\mathcal{F}$. Abusing notation, we write conditional probabilities as[1]

$$\begin{cases} p_{\mathcal{M}}(z_1 = e) := \mathbb{1}[\mathrm{src}(e) = s_0], \\ p_{\mathcal{M}}(z_i = e' \mid z_{i-1} = e) := \mathbb{1}[\mathrm{dst}(e) = \mathrm{src}(e')], \\ p_{\mathcal{M}}(x_i = v \mid z_i = e) := \mathbb{1}[v \in \mathrm{label}(e)]. \end{cases}$$

Marginalizing over the latent paths yields

$$p_{\mathcal{M}}(x_{1:L}) \propto \sum_{z_{1:L}} p_{\mathcal{M}}(z_1) \prod_{i=2}^{L} p_{\mathcal{M}}(z_i \mid z_{i-1}) \prod_{i=1}^{L} p_{\mathcal{M}}(x_i \mid z_i)\ \mathbb{1}[\mathrm{dst}(z_L) \in \mathcal{F}]. \tag{3}$$

The value of $p_{\mathcal{M}}(x_{1:L})$ is proportional to the number of accepting paths that generate $x_{1:L}$; if the finite automaton is *deterministic*, then each accepted sequence has exactly one accepting path, thus $p_{\mathcal{M}}$ becomes uniform over $\mathcal{C}$. The chain structure allows $p_{\mathcal{M}}$ and its marginals to be computed exactly via forward–backward message passing [20].

4 Method

4.1 Problem Statement

Our goal is to sample from a dLLM's generative distribution p_{dLLM} restricted to a constraint $\mathcal{C}$, i.e., $p_{\mathrm{dLLM}}(x^0 \mid x^0 \in \mathcal{C})$. However, sampling exactly from this distribution is intractable, and one naive resort is rejection sampling against the full reverse process, which is infeasible in practice due to extremely low acceptance rates. For autoregressive LLMs, a common practice is to mask out next tokens that would violate the constraint as generation proceeds left-to-right [34]. Following an analogous strategy, we enforce that samples drawn from *each diffusion step* must satisfy the constraint. Recall from Equation (2) that a standard diffusion step samples $\hat{x}^0$ from the model's mean-field distribution. We instead sample from this distribution *restricted to* $\mathcal{C}$, ensuring $\hat{x}^0 \in \mathcal{C}$ at every step, so the final output is guaranteed to satisfy the constraint by construction. Concretely, at each denoising step we draw

$$\hat{x}^0 \ \sim\ p_\theta(x^0 \mid x^t, \mathcal{C}) \ \propto\ \left(\prod_{i=1}^{L} p_\theta(x_i^0 \mid x^t) \right) \cdot \mathbb{1}[x^0 \in \mathcal{C}]. \tag{4}$$

Why this is challenging. At each diffusion step, the mean-field prediction $\prod_i p_\theta(x_i^0 \mid x^t)$ is fully factorized, i.e. each position is sampled independently, while the constraint $\mathcal{C}$ couples positions globally. For example, if $\mathcal{C}$ only accepts valid real numbers, both "`1.`" and "`.1`" are valid length-2 sequences, but a factorized sampler that draws "`.`" at position 1 (consistent with "`.1`") and "`.`" at position 2 (consistent with "`1.`") produces the invalid sequence "`..`". Therefore, the autoregressive remedy of filtering out invalid next tokens does not apply: the validity of position i depends on the tokens being sampled simultaneously at all other positions. Sampling exactly from Equation (4) therefore requires reasoning about the joint distribution induced by the constraint.

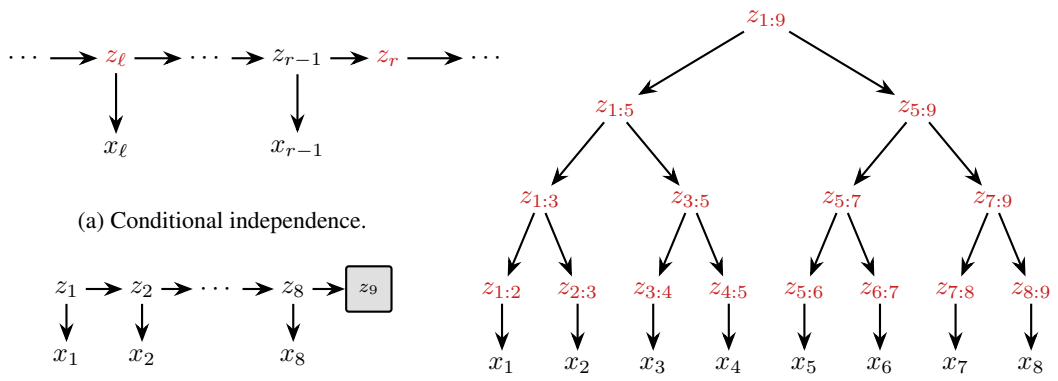
4.2 Tractable Inference over Finite Automata Constraints

Inference over general constraints is intractable, and for this work, we focus on constraints expressible by finite automata. As reviewed in Section 3.2, an automaton induces a chain-structured graphical model (Figure 1) and hence a distribution $p_{\mathcal{M}}(x^0_{1:L})$ over length-L sequences whose support is exactly the constrained language, i.e., $p_{\mathcal{M}}(x^0_{1:L}) > 0$ if and only if $x^0_{1:L} \in \mathcal{C}$.

As described in Section 3.2, if the finite automaton is deterministic then $p_{\mathcal{M}}$ is uniform over $\mathcal{C}$. Sampling from the constrained posterior in Equation (4) therefore reduces to sampling from the product of mean-field and constrained distribution:

$$\hat{x}^0 \ \sim\ \left(\prod_{i=1}^{L} p_\theta(x_i^0 \mid x^t) \right) \cdot p_{\mathcal{M}}(x^0_{1:L}) \ \propto\ \text{Equation (4)} \tag{5}$$

[1] These are unnormalized factors rather than conditional probabilities; we treat them as such and implicitly normalize globally over length-L sequences during sampling.



(a) Conditional independence.

(b) Chain-structured ($L = 8$).

(c) Tree-structured ($L = 8$)

Figure 2: Chain-structured graphical model and its tree-structured equivalent. **(a)** Given boundary states z_ℓ and z_r, $x_{\ell:r-1}$ is independent of the rest, the property exploited by the tree decomposition. **(b)** Naive sequential chain. **(c)** Tree-structured equivalent enabling $\mathcal{O}(\log L)$ parallel sampling.

which is exactly the desired constrained mean-field distribution. For nondeterministic finite automata (NFAs), both the product construction and sampling remain exact and tractable. However, the resulting distribution is a path-count-weighted version of the constrained posterior $p_\theta(x^0 \mid x^t, \mathcal{C})$, while still guaranteeing constraint satisfaction. Removing this path-count weighting is generally intractable, as counting distinct accepted sequences of a fixed length (the #NFA problem) is #P-complete [1].

Tractable product construction. Given two distributions p_θ and $p_\mathcal{M}$, our goal is to sample from $g(x^0_{1:L}) \propto p_\theta(x^0_{1:L}) \cdot p_\mathcal{M}(x^0_{1:L})$. The product of two arbitrary distributions is in general intractable. In our setting, however, p_θ is fully factorized (mean-field) and $p_\mathcal{M}$ is chain-structured, thus their product admits a tractable representation with the same chain structure as $p_\mathcal{M}$ in Figure 1 [26, 33].

Concretely, the product is obtained by reweighting each individual emission factor of $p_\mathcal{M}$ with the corresponding mean-field term of $p_\theta(x^0 \mid x^t) = \prod_{i=1}^{L} p_\theta(x^0_i \mid x^t)$, while leaving the transition factors unchanged. This yields a new chain-structured model (Figure 1(b)) with factors

$$\begin{cases} g(z_1 = e) := p_\mathcal{M}(z_1 = e), \\ g(z_i = e' \mid z_{i-1} = e) := p_\mathcal{M}(z_i = e' \mid z_{i-1} = e), \\ g(x_i = v \mid z_i = e) := p_\mathcal{M}(x_i = v \mid z_i = e) \cdot p_\theta(x_i = v \mid x^t) \end{cases}$$

Note that these reweighted factors are unnormalized and hence do not define conditional probabilities on their own; locally normalizing each factor would yield a distribution that is different from $g(x^0_{1:L})$. Instead, we normalize g globally over length-L sequences to preserve the original distribution. As in standard chain-structured graphical models [23], to sample from g, we first perform a backward pass to compute the required normalization constants for each factor [20] and then draw samples via ancestral sampling.

Time complexity. The backward message-passing costs $\mathcal{O}(L(|\mathcal{E}|^2 + |\mathcal{E}||\mathcal{V}|))$ FLOPs: the $\mathcal{O}(L|\mathcal{E}|^2)$ term comes from the message propagation from z_t to z_{t-1} and the $\mathcal{O}(L|\mathcal{E}||\mathcal{V}|)$ term arises from emission reweighting using LM probabilities. The forward pass adds $\mathcal{O}(L(|\mathcal{E}| + |\mathcal{V}|))$ FLOPs of work, sampling z_i and x_i for $1 \le i \le L$. In practice, we use the state-space formulation, reducing the total number of FLOPs to $\mathcal{O}(L(|\mathcal{S}|^2 + |\mathcal{E}||\mathcal{V}|))$ by performing message passing over the states of the finite automaton instead of the edges. Details in Appendix A.

Remasking confidence and diffusion sampling. dLLMs select which tokens to commit at each step using a per-position confidence score, typically the unconstrained marginal $p_\theta(x^0_i \mid x^t)$. Under a constraint $\mathcal{C}$, the appropriate score is the constrained marginal $p_\theta(x^0_i = \hat{x}_i \mid x^t, \mathcal{C})$, which reflects how strongly the constraint and the model jointly prefer $\hat{x}_i$ at position i.

Algorithm 1 Parallel constrained sampling from $p_\theta(x^0 \mid x^t, \mathcal{C})$

Require: Diffusion predictions $p_\theta(x^0 \mid x^t)$, constraint $\mathcal{C}$, length L
Ensure: Sample $x^0 \sim p_\theta(x^0 \mid x^t, \mathcal{C})$

for $k = 0, 1, \ldots, \log_2 L$ **do** ▷ compute messages bottom-up
  Compute 2^k-step transitions $\{p(z_{i+2^k} \mid z_i)\}_i$ from 2^{k-1}-step transitions
end for
Sample boundary states (z_1, z_{L+1}) from their joint marginal ▷ sample top-down
return SAMPLE$(1, L+1, z_1, z_{L+1})$

procedure SAMPLE(ℓ, r, z_ℓ, z_r)
  if $r - \ell \leq 1$ **then**
    return sample $x_\ell \sim p(x_\ell \mid z_\ell, z_r)$ ▷ base case
  end if
  $m \leftarrow \lfloor (\ell + r)/2 \rfloor$
  Sample $z_m \sim p(z_m \mid z_\ell, z_r) \propto p(z_m \mid z_\ell) \cdot p(z_r \mid z_m)$
  in parallel:
    $x_{\ell:m-1} \leftarrow$ SAMPLE(ℓ, m, z_ℓ, z_m)
    $x_{m:r-1} \leftarrow$ SAMPLE(m, r, z_m, z_r)
  return concat$(x_{\ell:m-1}, x_{m:r-1})$
end procedure

In diffusion models, this sampling procedure must be repeated at every denoising step; as a result, the $\mathcal{O}(L)$ sequential depth compounds across steps and becomes a major bottleneck. To address this, we introduce a log-depth sampling algorithm that reduces the sequential dependence to $\mathcal{O}(\log L)$.

4.3 Efficient Log-depth Tree Sampling

Standard ancestral sampling, as well as forward/backward message passing, is sequential for the chain-structured graphical model in Figure 2b: each z_i depends on z_{i-1}, so we must proceed through all L states iteratively, even though the computation at each step is small (on the order of 10^6 FLOPs). To expose more parallelism and fully leverage modern GPUs, we apply depth-reduction techniques for arithmetic circuits [32, 38], reducing the number of sequential steps from $\mathcal{O}(L)$ to $\mathcal{O}(\log L)$.

The key observation is that, for any states $l < m < r$, conditioning on the two boundary states z_ℓ and z_r makes the tokens $x_{\ell:m-1}$ and $x_{m:r-1}$ conditionally independent given the midpoint state z_m [21]:

$$p(x_{\ell:r-1} \mid z_\ell, z_r) \;\propto\; \sum_{z_m} \underbrace{p(z_m \mid z_\ell, z_r)}_{\text{midpoint}} \cdot \underbrace{p(x_{\ell:m-1} \mid z_\ell, z_m)}_{\text{left half}} \cdot \underbrace{p(x_{m:r-1} \mid z_m, z_r)}_{\text{right half}}, \quad \ell < m < r \tag{6}$$

where we typically take $m = (\ell + r)/2$. For example, in Figure 2c, conditioning on z_1 and z_9 makes $x_{1:4}$ and $x_{5:8}$ conditionally independent given z_5. This enables a divide-and-conquer sampling algorithm. Instead of sampling the latent states sequentially as $z_1, x_1, z_2, x_2, \ldots, z_L, x_L$, we first sample the midpoint latent state $z_{\lfloor L/2 \rfloor + 1}$ given the boundary states. The left and right subproblems are then conditionally independent and can therefore be sampled in parallel. Recursively applying this procedure yields a sampler of depth $\log L$, with all subproblems at each recursion level processed in parallel (Figure 2c). The base case samples all tokens in parallel from the emission factors $p(x_i \mid z_i, z_{i+1}) \propto p(x_i \mid z_i) \cdot p(z_{i+1} \mid z_i)$. The conditional coefficients $p(z_m \mid z_\ell, z_r)$ in Equation (6) are not given by the model directly, but can be computed by recursion as well from the relation:

$$p(z_m \mid z_\ell, z_r) \;\propto\; p(z_m \mid z_\ell) \cdot p(z_r \mid z_m), \tag{7}$$

where the right-hand side are multi-step transition probabilities, and all of which can be computed in $\log L$ steps. The full algorithm consists of two phases: a *bottom-up* pass that precomputes the multi-step transitions, and a *top-down* divide-and-conquer pass that samples midpoint states level by level. Once all hidden states $z_{1:L}$ are sampled, the tokens $x_{1:L}$ are sampled independently in parallel from $p(x_i \mid z_i, z_{i+1})$. Pseudocode is given in Algorithm 1; the proof is in Appendix B.

Time complexity. The bottom-up parallel pass computes all multi-step transitions via Equation (7), costing $\mathcal{O}(L(|\mathcal{E}|^3 + |\mathcal{E}||\mathcal{V}|))$ FLOPs in $\mathcal{O}(\log L)$ depth: the $\mathcal{O}(L|\mathcal{E}|^3)$ term comes from $|\mathcal{E}| \times |\mathcal{E}|$

matrix multiplications, while the $\mathcal{O}(L|\mathcal{E}||\mathcal{V}|)$ emission reweighting term is the same as in ancestral sampling. The top-down sampling pass samples the hidden states $z_{1:L+1}$ in $\mathcal{O}(L|\mathcal{E}|)$ work and $\mathcal{O}(\log L)$ depth via the recursion tree, after which all tokens $x_{1:L}$ are sampled fully in parallel at $\mathcal{O}(L|\mathcal{V}|)$ work. Similar to ancestral sampling, the total number of FLOPs can be reduced to $\mathcal{O}(L(|\mathcal{S}|^3 + |\mathcal{E}||\mathcal{V}|))$ in $\mathcal{O}(\log L)$ depth when performing message passing over states.

Parallelization vs. FLOPs tradeoff. Compared to ancestral sampling, log-depth sampling increases the message-passing FLOPs by a factor of $|\mathcal{S}|$, but reduces the depth of the computation graph from $\mathcal{O}(L)$ to $\mathcal{O}(\log L)$. This tradeoff is favorable on modern GPUs: the additional FLOPs are spent on larger matrix operations, while the number of sequential GPU calls is reduced from $\mathcal{O}(L)$ to $\mathcal{O}(\log L)$. Since each step of ancestral sampling performs relatively little work, its runtime is often dominated by synchronization and kernel-launch overhead, while the log-depth sampler exposes substantially more parallelism at each layer, leading to higher GPU utilization and lower wall-clock latency.

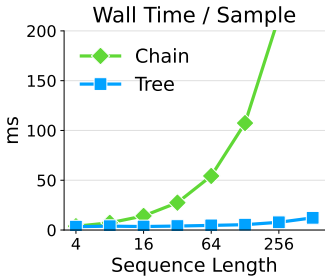


Figure 3: Runtime comparison for chain (ancestral) and tree (log-depth) sampling on a DFA.

5 Experiments

5.1 Tasks, Models and Setups

Tasks We evaluate our method on five tasks across six datasets.
Constraints are encoded as finite automata (DFA or NFA); full prompts and example outputs are provided in Appendix C. [2]

- **Function Calling (xLAM, BFCL).** *Task:* Given a user query and a set of available function schemas, generate the corresponding function call. *Data:* xlam-function-calling-60k (xLAM) [11] and the single-turn split of the Berkeley Function Calling Leaderboard (BFCL) [19], including both non-live and live settings. *Metric:* Exact match against the ground-truth function call. *Constraint:* A DFA enforcing that the output conforms to the given function schema. We evaluate two output formats: *Python format* (the BFCL standard, e.g., `get_weather(location="NYC")`) and *JSON format* (e.g., `{"name": "get_weather", "location": "NYC"}`).
- **Sudoku.** *Task:* Solve a 4×4 Sudoku puzzle [35]. *Metric:* Exact match against valid Sudoku solutions. Averaged across number of prefilled cells $\{4, \ldots, 12\}$. *Constraint:* A DFA enforcing the 4×4 output format and preserving prefilled input cells; row, column, and box constraints are not enforced by the DFA, which would require an exponentially larger DFA.
- **Countdown.** *Task:* Reach a target number using arithmetic operations on three given numbers [35]. *Metric:* Whether the expression evaluates to the target. *Constraint:* A DFA enforcing A op $B = C$ at each step, with operands from the input or prior results and the final result equal to the target.
- **Math reasoning (GSM-Symbolic).** *Task:* Solve grade-school math word problems. *Data:* GSM-Symbolic [13], where numerical values and names are replaced by symbolic variables. *Metric:* Exact match on the final answer. *Constraint:* A DFA enforcing that symbolic expressions appear only within « and », while the remaining text is unrestricted natural language.
- **Text-to-SQL (Spider).** *Task:* Given a user query and a database schema, generate the corresponding SQL query. *Data:* Spider [37]. *Metric:* Execution accuracy. A query is correct if it returns the same results as the ground truth on the test database. *Constraint:* An NFA encodes SQL grammar restricted to the columns and tables in the provided schema.

Models We evaluate two open-weight dLLMs, LLaDA-8B [15] and Dream-7B [36], each with *Base* and *Instruct* variants. We use the *Base* variants for Sudoku and Countdown and the *Instruct* variants for the remaining tasks [36]. For position selection, LLaDA selects the highest-probability positions, while Dream selects the lowest entropy ones. Our method computes marginal probabilities under the constrained distribution at each step and uses them as the confidence signal for both models. For block-wise decoding, we enforce the constraint globally over the entire sequence, while only committing sampled tokens within the current block.

[2] Code is available at https://github.com/MhDang/mosaic.

Table 1: Comparing unconstrained (Base) and constrained (Ours) accuracy (%) under greedy decoding and stochastic sampling on Dream-7B and LLaDA-8B. CS is the baseline constraint satisfaction rate; our method guarantees 100% constraint satisfaction by construction. For function calling, we report both *JSON* and *Python* formats; for BFCL, L/NL denote Live and Non-Live settings.

	Dream-7B						**LLaDA-8B**					
	Greedy			**Sampling**			**Greedy**			**Sampling**		
Task / Method	CS	Base	Ours	CS	Base	Ours	CS	Base	Ours	CS	Base	Ours
Function Calling												
xLAM *(JSON)*	90.1	74.6	**76.6**	59.2	50.5	**75.7**	90.2	67.8	**69.0**	86.4	64.2	**68.4**
BFCL L *(JSON)*	86.7	63.9	**71.5**	31.2	22.3	**69.0**	89.5	59.6	**65.6**	84.7	56.6	**63.3**
BFCL NL *(JSON)*	90.3	79.9	**82.4**	47.1	40.8	**82.6**	93.5	81.7	**85.5**	92.9	80.5	**82.8**
xLAM *(Python)*	70.3	56.1	**76.4**	25.6	20.9	**76.8**	77.3	60.1	**71.4**	69.4	54.2	**70.8**
BFCL L *(Python)*	36.0	22.4	**69.7**	4.4	2.8	**67.8**	74.5	50.3	**64.0**	68.3	46.8	**63.0**
BFCL NL *(Python)*	44.4	36.3	**78.3**	8.4	7.2	**76.2**	83.8	71.4	**83.8**	76.1	62.3	**81.4**
Planning												
Sudoku	38.0	36.4	**92.4**	7.6	4.7	**91.9**	20.7	19.8	**79.6**	19.8	18.7	**77.9**
Countdown	53.6	53.6	**100.**	35.2	35.2	**100.**	53.8	53.8	**100.**	53.5	53.5	**100.**
Code & Math												
Spider	72.2	52.7	**53.0**	19.3	15.5	**52.1**	73.5	53.2	**53.6**	69.1	49.8	**53.3**
GSM-Symbolic	80.2	39.9	**40.2**	54.6	15.0	**38.0**	81.5	**49.6**	49.2	80.1	46.9	**49.2**

5.2 Main Results

Table 1 reports baseline and constrained accuracy across five tasks under two decoding regimes: greedy decoding (temperature 0) and stochastic sampling (temperature 1), along with the baseline's constraint satisfaction rate. Our method guarantees 100% constraint satisfaction by construction.

Stochastic sampling collapses unconstrained dLLMs; constrained decoding recovers near-greedy accuracy. Without constraints, switching from greedy ($T{=}0$) to stochastic decoding ($T{=}1$) destroys structured-output performance: Dream's BFCL-Live (JSON) accuracy drops from 63.9% to 22.3%. The collapse is largely driven by constraint satisfaction: under sampling, far more outputs fail to satisfy. Our method improves greedy accuracy across tasks and, more interestingly, preserves accuracy under stochastic sampling where the unconstrained baseline collapses (69.0%, vs. 22.3% baseline), largely closing the gap to greedy performance.

The format gap is structural, not semantic. Switching from JSON to Python format sharply degrades baseline performance (Dream BFCL-Live: 63.9% to 22.4%; LLaDA: 59.6% to 50.3%), suggesting that these models know *what function* to call but struggle with *how* to format it. Our method largely closes this gap (Dream: 71.5% to 69.7%; LLaDA: 65.6% to 64.0%) by supplying the missing structure, enabling generalization across formats.

Improvements come from both filtering invalid sequences and remasking under the constrained distribution. We compare two remasking strategies. (Mf) uses the baseline's mean-field prediction as confidence to select positions. (Mar) recomputes confidence as marginal probabilities under the constrained distribution, changing the decoding order itself. Table 2 shows that Mar outperforms Mf, with the gap largest when the unconstrained baseline performs poorly. This pattern suggests "filtering" alone cannot find high-probability valid sequences when the model's mean-field is misaligned with the constraint; recomputing marginals provides the missing signal.

5.3 Ablation and Analysis

Runtime overhead is small relative to accuracy gains. We measure per-sample latency, throughput, and accuracy on a 200-example subset of BFCL. As shown in Table 3, our tree variant improves accuracy by 12 points over the unconstrained baseline while incurring only 4% wall-clock overhead. This shows that finite automaton inference is lightweight compared to dLLM and parallelizes well.

Table 2: Ablation on remasking. Mf uses base model's mean-field prediction as confidence; Mar uses marginals under constrained distribution as confidence.

	Dream			LLaDA		
Task	Base	Mf	Mar	Base	Mf	Mar
xLAM *(JSON)*	74.6	**77.5**	76.6	67.8	**69.1**	69.0
xLAM *(Python)*	56.1	68.4	**76.4**	60.1	58.7	**71.4**
BFCL-Live *(JSON)*	63.9	69.4	**71.5**	59.6	64.9	**65.6**
BFCL-Live *(Python)*	22.4	61.5	**69.7**	50.3	61.5	**64.0**

Table 3: Runtime, accuracy, and throughput scaling on 200 BFCL examples (Dream). Latency uses batch size 1; throughput reports tokens/s as batch size increases. Measured on an A6000. Overhead is wall-clock relative to the unconstrained baseline. Mean ± std over 3 seeds.

Method	**Accuracy**	**Latency**		**Throughput (tok/s)**		
		s/sample	**Overhead**	**bs=1**	**bs=2**	**bs=4**
Unconstrained	67.5	24.26 ± 0.05	—	10.55 ± 0.02	10.79 ± 0.06	11.17 ± 0.05
Ours (chain)	78.5 ± 0.2	51.92 ± 0.14	+114%	4.93 ± 0.01	6.88 ± 0.06	8.62 ± 0.05
Ours (tree)	79.2 ± 0.5	25.35 ± 0.04	+4%	10.10 ± 0.02	10.21 ± 0.02	10.44 ± 0.03

Scaling with fewer generation steps. Figure 4 evaluates Dream and LLaDA on the BFCL simple split as denoising steps decrease from 256 to 16. Baseline performance drops sharply with fewer steps, whereas our method remains much more robust. With more parallelism, unconstrained sampling produces more invalid tokens, whereas our method maintains valid structure and preserves accuracy.

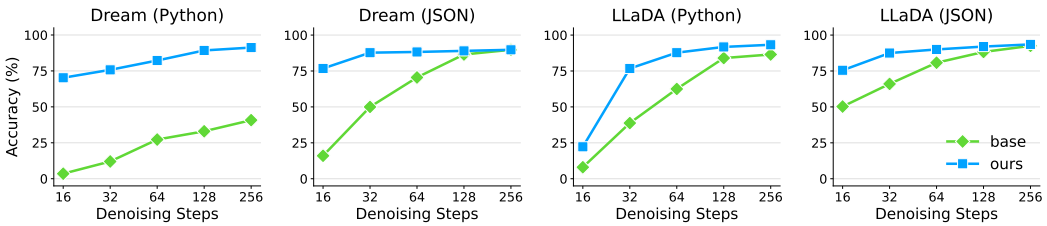


Figure 4: Accuracy vs. denoising steps on BFCL Simple split. Our method remains robust as the number of denoising steps decreases, whereas the unconstrained baseline deteriorates sharply.

Comparison with dLLMs and autoregressive LLMs of similar size. Table 4 compares diffusion LLMs with similarly sized autoregressive models on BFCL. Unconstrained diffusion models perform substantially worse and our method substantially narrows the gaps.

Table 4: BFCL accuracy (%) comparing diffusion LLMs and autoregressive LLMs (Python format), with autoregressive results from BFCL leaderboard [19]. Our method substantially narrows the gap.

	Diffusion LLMs (dLLMs)				**Autoregressive LLMs**	
Setting	Dream	Dream + Ours	LLaDA	LLaDA + Ours	Llama-3.1-8B	Qwen3-8B
Non-live	36.3	**78.3**	71.4	**83.8**	84.0	88.6
Live	22.4	**69.7**	50.3	**64.0**	70.8	80.1

NFAs vs. DFAs. The constrained decoding workload spans over three orders of magnitude, from a 21-state *DFA* for Sudoku, to DFAs with ∼1k states for function calling, and *NFAs* with ∼10k states and ∼100k edges for Spider. We use DFAs when their sizes are feasible and fall back to NFAs for Spider, where the equivalent DFA would be exponentially large (Details in Appendix C).

Comparison with DINGO. We compare our method with DINGO [30] on GSM-Symbolic, using the decoding configuration and published results from their paper. As shown in Table 5, our method achieves higher accuracy under greedy decoding, while additionally supporting stochastic sampling, which DINGO does not support.

Table 5: Accuracy comparison with DINGO on GSM-Symbolic.

	Greedy		Sampling	
Method	Dream	LLaDA	Dream	LLaDA
DINGO ($d = 16$)	36.0	32.0	Not supported	
Ours ($d = 16$)	36.6	46.6	–	–
Ours (default d)	**40.2**	**49.2**	**38.0**	**49.2**

Comparison with rejection sampling. We evaluate rejection sampling on 200 examples from BFCL Non-Live (JSON), with a maximum budget of 16 draws per example. When single-draw constraint satisfaction is low, rejection sampling may require many attempts and fail to find a valid output within the budget. As shown in Table 6, rejection sampling fails on 8.5% and 1.0% of examples for Dream and LLaDA, respectively, while requiring an average of 3.94 and 1.40 draws to obtain a valid output. In contrast, our method achieves comparable or even higher accuracy while guaranteeing constraint satisfaction by construction.

Table 6: Comparison with rejection sampling on BFCL Non-Live (JSON), evaluated on 200 examples with a maximum budget of 16 draws.

Metric	Dream-7B	LLaDA-8B
Accuracy (%)		
Rejection sampling	78.14	**81.81**
Ours	**81.33**	81.17
Rejection Sampling Statistics		
Single-draw constraint satisfaction (%)	45.06	87.97
Avg. draws until first valid sample	3.94	1.40
Failure within 16 draws (%)	8.5	1.0

6 Conclusion

We propose a constrained decoding algorithm for diffusion language models under constraints expressible as finite automata. Viewing finite automata as graphical models, our approach performs exact and tractable sampling/inference over the constrained mean-field posterior at each denoising step, guaranteeing constraint satisfaction. We further propose a log-depth sampling strategy to unlock greater parallelization, yielding low overhead in practice. Our approach substantially improves model performance across a variety of downstream tasks where unconstrained models struggle.

7 Limitations and Societal Impacts

Our method focuses on finite automata, and does not cover richer constraints such as context-free grammars. Constrained decoding can improve the reliability of LLM systems used for function calling and other structured-output applications. However, if misused, this method could enable the generation of syntactically valid but malicious tool calls or database queries, increasing the risk of unintended or harmful system actions.

Acknowledgments

This research is supported in part by ARO (W911NF-21-1-0125), ONR (N00014-23-1-2159), and the CZ Biohub.

References

[1] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. #NFA admits an FPRAS: Efficient enumeration, counting, and uniform generation for logspace classes. *Journal of the ACM*, 68(6):1–40, 2021.

[2] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*, volume 34, pages 17981–17993, 2021.

[3] Michael Cardei, Jacob K. Christopher, Bhavya Kailkhura, Thomas Hartvigsen, and Ferdinando Fioretto. Constrained discrete diffusion. In *Advances in Neural Information Processing Systems*, 2025.

[4] 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 *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

[5] Yixin Dong, Charlie F Ruan, Yaxing Cai, Ziyi Xu, Yilong Zhao, Ruihang Lai, and Tianqi Chen. XGrammar: Flexible and efficient structured generation engine for large language models. In *Proceedings of Machine Learning and Systems*, volume 7, 2025.

[6] Google DeepMind. Gemini diffusion. https://blog.google/innovation-and-ai/models-and-research/google-deepmind/gemini-diffusion/, 2025. Google AI Blog.

[7] Guidance AI. Guidance: A guidance language for controlling large language models. https://github.com/guidance-ai/guidance, 2023.

[8] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson, 3 edition, 2006.

[9] Inception Labs, Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, Aditya Grover, and Volodymyr Kuleshov. Mercury: Ultra-Fast Language Models Based on Diffusion. *arXiv preprint arXiv:2506.17298*, 2025.

[10] Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *International Conference on Learning Representations*, 2025.

[11] Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. APIGen: Automated pipeline for generating verifiable and diverse function-calling datasets. In *Advances in Neural Information Processing Systems*, 2024.

[12] Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

[13] Seyed Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-Symbolic: Understanding the limitations of mathematical reasoning in large language models. In *International Conference on Learning Representations*, 2025.

[14] Niels Mündler, Jasper Dekoninck, and Martin Vechev. Constrained decoding of diffusion LLMs with context-free grammars. In *International Conference on Learning Representations*, 2026.

[15] 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 *Advances in Neural Information Processing Systems*, 2025.

[16] Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. In *International Conference on Learning Representations*, 2025.

[17] Kanghee Park, Timothy Zhou, and Loris D'Antoni. Flexible and efficient grammar-constrained decoding. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

[18] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems*, 2024.

[19] Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The Berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

[20] Judea Pearl. Reverend Bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1982.

[21] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[22] Gabriel Poesia, Alex Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. Synchromesh: Reliable code generation from pre-trained language models. In *International Conference on Learning Representations*, 2022.

[23] Lawrence Rabiner and Biinghwang Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, 3(1), 1986.

[24] Subham Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems*, volume 37, pages 130136–130184, 2024.

[25] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

[26] Yujia Shen, Arthur Choi, and Adnan Darwiche. Tractable operations for arithmetic circuits of probabilistic models. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

[27] Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. In *Advances in Neural Information Processing Systems*, volume 37, pages 103131–103167, 2024.

[28] Richard Shin, Christopher Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. Constrained language models yield few-shot semantic parsers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

[29] Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, et al. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025.

[30] Tarun Suresh, Debangshu Banerjee, Shubham Ugare, Sasa Misailovic, and Gagandeep Singh. DINGO: Constrained inference for diffusion LLMs. In *Advances in Neural Information Processing Systems*, 2025.

[31] Shubham Ugare, Tarun Suresh, Hangoo Kang, Sasa Misailovic, and Gagandeep Singh. SynCode: LLM generation with grammar augmentation. *Transactions on Machine Learning Research*, 2025.

[32] L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM Journal on Computing*, 12(4):641–644, 1983.

[33] Antonio Vergari, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck. A compositional atlas of tractable circuit operations for probabilistic inference. In *Advances in Neural Information Processing Systems*, 2021.

[34] Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*, 2023.

[35] Jiacheng Ye, Shansan Gong, Liheng Chen, Lin Zheng, Jiahui Gao, Han Shi, Chuan Wu, Xin Jiang, Zhenguo Li, Wei Bi, and Lingpeng Kong. Diffusion of thought: Chain-of-thought reasoning in diffusion language models. In *Advances in Neural Information Processing Systems*, 2024.

[36] 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.

[37] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.

[38] Honghua Zhang, Benjie Wang, Marcelo Arenas, and Guy Van den Broeck. Restructuring tractable probabilistic circuits. In *International Conference on Artificial Intelligence and Statistics*, 2025.

[39] Yitong Zhang, Yongmin Li, Yuetong Liu, Jia Li, Xiaoran Jia, Zherui Li, and Ge Li. Lookahead-then-Verify: Reliable constrained decoding for diffusion LLMs under context-free grammars. *Proceedings of the ACM on Software Engineering*, 3(ISSTA):1264–1286, 2026.

[40] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024.

A Forward-Backward Inference for Chain-Structured Graphical Models

We give the standard forward–backward algorithm for the chain-structured graphical model defined in Section 3.2 (Figure 1). We first present the *edge-space* formulation, which mirrors the edge-valued latents $z_t \in \mathcal{E}$ used in the main text, and then aggregate edge messages by their endpoint state to obtain the equivalent *state-space* formulation, which has lower cost. This is a reformulation of the standard hidden Markov models backward message passing and sampling algorithm [23].

A.1 Edge Space

We use the unnormalized factors of Section 3.2:

$$\begin{cases} g(x_i = v \mid z_i = e) := p_{\mathcal{M}}(x_i = v \mid z_i = e) \cdot p_\theta(x_i = v \mid x^t), \\ g(z_1 = e) := p_{\mathcal{M}}(z_1 = e), \\ g(z_i = e' \mid z_{i-1} = e) := p_{\mathcal{M}}(z_i = e' \mid z_{i-1} = e). \end{cases}$$

We sample from g in two passes.

Backward pass. Define backward messages

$$\beta_t : \mathcal{E} \to \mathbb{R}_{\geq 0} \quad \text{for} \quad t = 1, \ldots, L,$$

where $\beta_t(e)$ is the partition function of the suffix $(z_{t+1:L}, x_{t+1:L})$ under g given $z_t = e$ restricted to constraint-satisfying sequences. With the base case absorbing the accept condition

$$\beta_L(e) \;=\; \mathbb{1}[\mathrm{dst}(e) \in \mathcal{F}],$$

the recurrence for $t = L{-}1, L{-}2, \ldots, 1$ is

$$\beta_t(e) \;=\; \sum_{e' \in \mathcal{E}} g(z_{t+1}{=}e' \mid z_t{=}e) \left(\sum_{v \in \mathcal{V}} g(x_{t+1}{=}v \mid z_{t+1}{=}e') \right) \beta_{t+1}(e').$$

The structural constraints live inside g: $g(z_{t+1} \mid z_t)$ vanishes unless the edges chain end-to-end, and $g(x_{t+1} \mid z_{t+1})$ vanishes outside the edge label set, so the sums above effectively range over chained successors and label-compatible tokens.

Forward sampling pass. We start by sampling (z_1, x_1) jointly from

$$\mathbb{P}\big(z_1 = e,\ x_1 = v\big) \;\propto\; g(z_1 = e)\, g(x_1 = v \mid z_1 = e)\, \beta_1(e). \tag{8}$$

For each subsequent step $t = 2, \ldots, L$, conditional on the previously sampled z_{t-1}, jointly sample (z_t, x_t) with probability

$$\mathbb{P}\big(z_t = e,\ x_t = v\big) \;\propto\; g(z_t = e \mid z_{t-1})\, g(x_t = v \mid z_t = e)\, \beta_t(e), \tag{9}$$

with the convention $g(z_1 \mid z_0) := g(z_1)$ for the initial step. The factor $\beta_t(e)$ reweights each candidate edge by the probability that the suffix from $\mathrm{dst}(e)$ extends into an accepting state. By construction, the resulting $x_{1:L}$ is the marginal of g/Z where Z is the partition restricted to accepting paths.

Time complexity. The backward message-passing pass costs $\mathcal{O}(L(|\mathcal{E}|^2 + |\mathcal{E}||\mathcal{V}|))$ FLOPs: the $\mathcal{O}(L|\mathcal{E}|^2)$ term comes from edge-to-edge message propagation, while the $\mathcal{O}(L|\mathcal{E}||\mathcal{V}|)$ term arises from emission reweighting using LM probabilities. The forward pass for sampling z_i and x_i with $1 \leq i \leq L$ adds $\mathcal{O}(L(|\mathcal{E}| + |\mathcal{V}|))$ FLOPs of work.

A.2 State Space

The same chain admits an equivalent presentation over states. Aggregating edges by their endpoints yields state-valued latents $s_t \in \mathcal{S}$ for $t = 0, 1, \ldots, L$ (with s_0 pinned to the FA's initial state), and the joint factors of g become

$$\begin{cases} g(s_0) \;=\; \mathbb{1}[s_0 = s_0^{\mathrm{init}}], \\ g(s_t, x_t = v \mid s_{t-1}) \;=\; \displaystyle\sum_{e:\, \mathrm{src}(e) = s_{t-1},\, \mathrm{dst}(e) = s_t} g(x_t = v \mid z_t = e), \end{cases}$$

together with the accept indicator $\mathbb{1}[s_L \in \mathcal{F}]$. The joint transition factor aggregates contributions from all edges that traverse the same state pair, naturally handling NFAs where multiple edges may connect s_{t-1} to s_t. It is convenient to define the per-position transition matrix

$$M_t \in \mathbb{R}_{\geq 0}^{|\mathcal{S}| \times |\mathcal{S}|}, \qquad M_t(s, s') \;:=\; \sum_{v \in \mathcal{V}} g(s_t = s',\, x_t = v \mid s_{t-1} = s),$$

which marginalizes over the emitted token.

Backward pass. Define $\beta_t : \mathcal{S} \to \mathbb{R}_{\geq 0}$ for $t = 0, 1, \ldots, L$ (overloading the symbol; the argument type disambiguates state from edge). With

$$\beta_L(s) \;=\; \mathbb{1}[s \in \mathcal{F}], \qquad \beta_{t-1}(s) \;=\; \sum_{s' \in \mathcal{S}} M_t(s, s')\, \beta_t(s'),$$

this is the standard HMM backward recurrence with transition matrix M_t, related to the edge-space messages by $\beta_t(s) = \beta_t(e)$ for any edge e with $\mathrm{dst}(e) = s$ (well-defined since edge-space $\beta_t(e)$ depends on e only through $\mathrm{dst}(e)$).

Forward sampling pass. Initialize $s_0 = s_0^{\text{init}}$. For each $t = 1, 2, \ldots, L$, jointly sample (s_t, x_t) with probability

$$\mathbb{P}(s_t = s',\, x_t = v \mid s_{t-1}) \;\propto\; g(s_t = s',\, x_t = v \mid s_{t-1})\, \beta_t(s').$$

Time complexity. Each backward update is $\mathcal{O}(|\mathcal{S}|^2)$, forming M_t over the vocabulary costs $\mathcal{O}(|\mathcal{E}|\,|\mathcal{V}|)$ per position, and forward sampling at each position contributes $\mathcal{O}(|\mathcal{S}| + |\mathcal{V}|)$ via the staged (s_t, x_t) draw described above. The total state-space cost is $\mathcal{O}\big(L\,(|\mathcal{S}|^2 + |\mathcal{E}|\,|\mathcal{V}|)\big)$, improving over the naive $\mathcal{O}(L\,|\mathcal{E}|^2)$ edge-space cost.

B Parallel Sampling Algorithm

We show that Algorithm 1 produces samples distributed exactly as the constrained posterior $p(x^0 \mid x^t, \mathcal{C})$, despite using unnormalized conditionals during sampling.

Proposition B.1. *Let $p(x^0 \mid x^t)$ denote the diffusion model's mean-field predictions, and let $\mathcal{C}$ be a constraint encoded as a deterministic finite automaton with state space $\mathcal{S}$. Algorithm 1 returns a sample x^0 distributed exactly as*

$$p(x^0 \mid x^t, \mathcal{C}) \;\propto\; p(x^0 \mid x^t) \cdot \mathbb{1}[x^0 \in \mathcal{L}(\mathcal{C})].$$

Proof. We proceed by induction on the recursion depth.

Base case. At a leaf of the recursion (segment with boundary states z_ℓ, z_r satisfying $r - \ell \leq 1$), the algorithm samples a single token x_ℓ from the conditional $p(x_\ell \mid z_\ell, z_r)$. Since x_ℓ ranges over a finite vocabulary, this categorical distribution can be normalized exactly by summing over its support, and the resulting sample is exact.

Inductive step. Consider a segment $[\ell, r]$ with boundary states z_ℓ, z_r and midpoint m. The algorithm performs three steps:

1. Sample z_m from $p(z_m \mid z_\ell, z_r) \propto p(z_m \mid z_\ell) \cdot p(z_r \mid z_m)$.
2. Recursively sample $x_{\ell:m-1}$ from the left subproblem with endpoints (z_ℓ, z_m).
3. Recursively sample $x_{m:r-1}$ from the right subproblem with endpoints (z_m, z_r).

Step 1 is a categorical sampling problem over the finite state space $\mathcal{S}$, and the unnormalized scores can be summed and normalized exactly in $\mathcal{O}(|\mathcal{S}|)$. By the inductive hypothesis, steps 2 and 3 produce exact samples from their respective conditional distributions. By d-separation in the hidden Markov model induced by $p(x^0 \mid x^t, \mathcal{C})$ (Figure 1), conditioning on z_ℓ, z_m, z_r renders $x_{\ell:m-1}$ and $x_{m:r-1}$ independent. Hence the joint distribution over the segment factors as

$$p(x_{\ell:r-1}, z_m \mid z_\ell, z_r) \;=\; p(z_m \mid z_\ell, z_r) \cdot p(x_{\ell:m-1} \mid z_\ell, z_m) \cdot p(x_{m:r-1} \mid z_m, z_r),$$

which is precisely the distribution sampled by steps 1–3. Thus the segment-level sample is exact.

Composition. The boundary states z_1, z_{L+1} at the root of the recursion are sampled from their joint marginal $p(z_1, z_{L+1})$, which is computed exactly from the precomputed skip-transitions. Combined with the inductive argument above, the full sample x^0 is distributed exactly as $p(x^0 \mid x^t, \mathcal{C})$.

Remark on unnormalized conditionals. Although each conditional $p(z_m \mid z_\ell, z_r)$ is written up to a (z_ℓ, z_r)-dependent normalization constant, this poses no issue for exactness: at sampling time, the constant is computed by summing the $|\mathcal{S}|$ unnormalized scores, and the resulting categorical distribution is exact. Different segments use different normalization constants, as they should—each subproblem is conditioned on different endpoint values, and the chain-rule decomposition above ensures these segment-local normalizers compose into the correct joint distribution. □

C Experiment Details

C.1 Generation Configurations

Table 7 summarizes the generation hyperparameters used for each task. We follow the default decoding configurations released with each model. Generation length and number of denoising steps are matched across models within each task to ensure a fair comparison.

Table 7: Generation settings per task across the evaluated diffusion language models. All runs use temperatures $T = 0$ and $T = 1$. The **Remasking** column reports the algorithm for both unconstrained and constrained decoding; for constrained runs, confidence values are recomputed from the constrained distribution.

Task	Model	Seq Length	Steps	Block Size	Remasking
BFCL *(Python / JSON)*	LLaDA-8B-Instruct	256	128	32	low_confidence
	Dream-v0-Instruct-7B	256	128	—	entropy
xLAM *(Python / JSON)*	LLaDA-8B-Instruct	128	64	32	low_confidence
	Dream-v0-Instruct-7B	128	64	—	entropy
Sudoku 4×4	LLaDA-8B-Base	32	32	32	low_confidence
	Dream-v0-Base-7B	32	32	—	entropy
Countdown *(cd3)*	LLaDA-8B-Base	32	32	32	low_confidence
	Dream-v0-Base-7B	32	32	—	entropy
GSM-Symbolic	LLaDA-8B-Instruct	128	64	32	low_confidence
	Dream-v0-Instruct-7B	128	64	—	entropy
Spider	LLaDA-8B-Instruct	128	64	32	low_confidence
	Dream-v0-Instruct-7B	128	64	—	entropy

C.2 Prompt Templates

We provide representative prompts and expected outputs for each evaluation task in Table 8.

Table 8: Representative prompts and expected outputs for each evaluation task. Repeated system prompts and in-context examples are omitted for space.

Task	Example Prompt	Example Answer

Function Calling (BFCL/ xLAM) *Python Format*	*System*: You are an expert in composing functions. You are given a question and a set of possible functions. Based on the question, you will need to make one or more function/tool calls to achieve the purpose.
	If none of the functions can be used, point it out. If the given question lacks the parameters required by the function, also point it out. You should only return the function calls in your response.
	If you decide to invoke any of the function(s), you MUST put it in the format of:
	`[func_name1(params_name1=params_value1, params_name2=params_value2, ...), func_name2(params)]`
	You SHOULD NOT include any other text in the response.
	At each turn, you should try your best to complete the tasks requested by the user within the current turn. Continue to output functions to call until you have fulfilled the user's request to the best of your ability. Once you have no more functions to call, the system will consider the current turn complete and proceed to the next turn or task.
	Here is a list of functions in JSON format that you can invoke:
	`{ "name": "calculate_triangle_area", "description": "Calculate the area of a triangle.", "parameters": { "base": integer, "height": integer } }`
	User: Find the area of a triangle with a base of 10 units and height of 5 units.
		`[calculate_triangle_area (base=10, height=5)]`

Function Calling (BFCL/ xLAM) *JSON Format*	*System: (...same as above...)* If you decide to invoke any of the function(s), you MUST put it in the format of a JSON array: `[{"name": "func_name1", "arguments": {"params_name1": params_value1, ...}}, ...]` *(...same as above...) User:* Find the area of a triangle with a base of 10 units and height of 5 units.	`[{"name": "calculate_triangle_area", "arguments": {"base": 10, "height": 5}}]`

Sudoku	Fill the positions where the values are 0 in a 4×4 grid with digits 1–4 so that each column, each row, and each of the four 2×2 subgrids that compose the grid contains all of the digits from 1 to 4.
	(In-context examples are omitted)
	Input:
	`1003`
	`0300`
	`3210`
	`4132`
	Output:
		`1423`
		`2341`
		`3214`
		`4132`

Countdown	Given 4 numbers, use $+, -, \times, \div$ to operate over the first three numbers to achieve the last one. *(In-context examples are omitted)* *Input:* `61, 83, 8, 30` *Output:*	`83-61=22, 8+22=30`

continued on next page

Table 8 continued from previous page

Task	Example Prompt	Example Answer

Math Reasoning (GSM-Symbolic)	*System:* You are an expert in solving grade school math tasks. You will be presented with a grade-school math word problem with symbolic variables and be asked to solve it.
	Before answering you should reason about the problem (using the `<reasoning>` field in the response described below). Intermediate symbolic expressions generated during reasoning should be wrapped in `« »`.
	Only output the symbolic expression wrapped in `« »` that answers the question. The expression must use numbers as well as the variables defined in the question. You are only allowed to use the following operations: `+, -, /, //, %, *, **`.
	You will always respond in the format described below:
	`Let's think step by step. <reasoning>`
	`The final answer is «symbolic expression»`
	(4 in-context examples are omitted)
	User: Rania saw a 210-foot whale with 7 72-inch remoras attached to it. What percentage of the whale's body length is the combined length of the remoras?
		Let's think step by step.
		The combined length of the remoras in inches is `«7 * 72»`.
		Convert to feet: `«7 * 72 / 12»`.
		Percentage: `«7 * 72 / 12 / 210 * 100»`.
		The final answer is `«7 * 72 / 12 / 210 * 100»`.

Text-to-SQL (Spider)	*System:* You are an expert in SQL. Given a database schema and a question, write a correct SQL query.
	User: Here is the database schema:
	`CREATE TABLE stadium (`
	`  stadium_id INT PRIMARY KEY,`
	`  location TEXT, name TEXT,`
	`  capacity INT, highest INT,`
	`  lowest INT, average INT`
	`);`
	(...more database omitted...)
	Question: How many singers do we have?
		`SELECT count(*)`
		`FROM singer`

C.3 Finite Automata Statistics

Table 9 reports the per-benchmark distribution of automaton sizes for the Dream-7B and LLaDA-8B tokenizers. For each benchmark we list the number of automata N used during evaluation and, across those N automata, the median, 95th percentile, and maximum of the node and edge counts. Reporting the median (rather than the mean) reflects the size of a typical instance, and the 95th percentile and maximum quantify the heavy right tail driven by a small number of large schemas.

Table 9: Automaton sizes across benchmarks (Dream-7B and LLaDA-8B tokenizers). All tasks use DFAs except Spider (NFA due to exponential blowup on DFA). We report number of automata N, state counts (tokenizer-independent), and edge counts for each tokenizer (median / P95 / max). GSM-Symbolic uses a single automaton.

		# States			**Edges (Dream)**			**Edges (LLaDA)**		
Task	N	Med	P95	Max	Med	P95	Max	Med	P95	Max
xLAM *JSON*	1000	154	387	1,400	542	1,539	7,414	507	1,415	6,550
xLAM *Python*	1000	102	287	1,227	397	1,280	6,986	374	1,222	6,154
BFCL NL *JSON*	1000	122	357	818	409	1,213	3,968	386	1,134	3,572
BFCL L *JSON*	1351	385	997	2,459	1,317	3,932	9,901	1,233	3,672	9,128
BFCL NL *Python*	1000	90	272	713	319	975	3,738	304	936	3,342
BFCL L *Python*	1351	301	778	1,613	1,112	2,839	7,576	1,042	2,694	6,704
Sudoku	900	21	21	21	21	21	21	21	21	21
Countdown	993	47	55	77	53	63	88	52	62	87
GSM-Symbolic	1	56	–	56	745	–	745	748	–	748
Spider	20	8,796	15,573	19,509	85,415	168,920	171,580	70,477	142,469	144,591

D Qualitative Examples on BFCL

Table 10 shows four representative cases on `live_simple`: in each one the model has the right intent but cannot keep its output inside the tool schema. Constraining decoding to the schema FA eliminates these failures by construction, recovering all four examples.

Table 10: Handling constraint violations on BFCL-Live (Dream-Instruct, T=0). The baseline often produces outputs that violate the tool specification, and receives 0% accuracy. E.g., prose instead of a tool call, incorrect casing, or hallucinated function names. Our constrained generation guarantees syntactically valid tool calls by construction (constraint satisfaction rate = 100%) and is also semantically correct on these examples.

(a) JSON: baseline emits prose instead of a tool call.

User *"I have a customer query asking for 'Need ATM location'. Which category should it be classified under?"*

Tool `record(atm_finder_start: list[str], ...)` – 6 enum-keyed list args.

Base `The customer query 'Need ATM location' should fall under the 'atm_finder_start' category.` (↪ Invalid format: not a JSON tool-call list.)

Ours `[{"name": "record", "arguments": {"atm_finder_start": ["Need ATM location"]}}]` **(correct)**

(b) JSON: wrong argument-name casing breaks the schema.

User *"Retrieve today's alerts for sensor Q3CC-CRT3-SZ2G, max* 10 *alerts per page."*

Tool `get_sensor_alerts(perPage: int*, sensorSerial: str*, ...)`

Base `[{"name": "get_sensor_alerts", "arguments": {"PerPage": 10, "t0": "2023-10-01", "sensorSerial": "Q3CC-CRT3-SZ2G"}}]` (↪ missing required 'perPage'; `t0` hallucinated.)

Ours `[{"name": "get_sensor_alerts", "arguments": {"perPage": 10, "sensorSerial": "Q3CC-CRT3-SZ2G"}}]` **(correct)**

(c) Python: hallucinated function name (dotted-method that doesn't exist in the tool list).

User *"I've completed the task 'Submit monthly financial report'. Mark it as completed."*

Tool `todo(type: enum["complete", ...], content: str)`

Base `[todo.complete(content="Submit monthly financial report")]` (↪ function name 'todo.complete' not in tool list.)

Ours `[todo(type="complete", content="Submit monthly financial report")]` **(correct)**

(d) Python: typo in argument name.

User *"Find me a Comic movie directed by Nitesh Tiwari."*

Tool `Movies_3_FindMovies(directed_by, genre, cast)`

Base `[Movies_3_FindMovies(ddirected_by="Nitesh Tiwari", genre="Comedy")]` (↪ unexpected parameter 'ddirected_by'.)

Ours `[Movies_3_FindMovies(directed_by="Nitesh Tiwari", genre="Comedy")]` **(correct)**

Constraining the output to the schema is not the same as constraining it to the gold answer. Table 11 shows three failures that survive our decoder: the schema accepts every output, but the value the model chose disagrees with the reference. These are the errors a token-level grammar cannot fix.

Table 11: Failure modes under constrained decoding on BFCL-Live (Dream-Instruct, T=0). All outputs satisfy the schema FA (valid format, function name, and argument types), but are still rejected by the official scorer because some argument values differ from the ground truth. These errors cannot be ruled out by token-level constraints: the model must choose among multiple schema-valid options and selects the wrong one.

(a) Over-specified location – *model copies the user's parenthetical landmark.*

User *"Tell me the current weather in Yosemite National Park, which locates at Mariposa, CA, in Celsius."*

Tool `get_current_weather(location: str, unit: str)`

Ours `[{"name": "get_current_weather", "arguments": {"location": "Yosemite National Park, Mariposa, CA", "unit": "celsius"}}]`

Gold `location = "Mariposa, CA"` (↪ schema OK; string-mismatch for `location`.)

(b) Hallucinated argument value – enum-legal but not the gold default.

User *"Set the air conditioner to cool mode."* (no wind speed, no temperature given)

Tool `ThinQ_Connect(body: dict)` with `body.windStrength` ∈ {LOW, MID, HIGH}.

Ours `[ThinQ_Connect(body={..., "windStrength": "HIGH", "coolTargetTemperature": 24, ...})]`

Gold `windStrength = "MID"`, no `targetTemperature`. (↪ FA accepts every enum value; LM picked the wrong one.)

(c) Missing optional-but-inferable argument.

User *"I made a $59.99 purchase in San Francisco, California. How much sales tax?"*

Tool `calculate_tax(purchase_amount, state, county, city, ...)`

Ours `[{"name": "calculate_tax", "arguments": {"purchase_amount": 59.99, "state": "CA"}}]`

Gold `state="CA", county="San Francisco", city="San Francisco"` (↪ `county` and `city` are schema-optional; LM omitted them.)

E Additional Experiment Results

Scaling with the difficulty of problems. In Sudoku, we vary the number of prefilled cells from 12 to 4, with fewer prefilled cells corresponding to more difficult puzzles. As shown in Figure 5, baseline performance drops sharply as the puzzles become harder. A common failure mode is that the model modifies prefilled cells, violating the puzzle constraints. Enforcing that these cells remain fixed effectively reduces the task to a text-infilling problem, leading to substantial improvements. Confidence-based scheduling further boosts performance by guiding decoding toward more likely valid solutions, particularly on the hardest puzzles where the base model struggles.

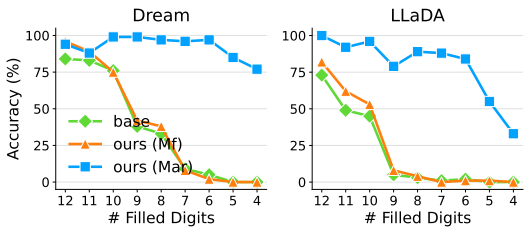


Figure 5: Sudoku 4×4 accuracy versus the number of prefilled digits (fewer digits indicate harder puzzles). Mf uses mean-field confidence from the base model, while Mar uses marginals under the constrained distribution.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state our contributions, an efficient constrained decoding algorithm for dLLM in Section 4, and experiments results in Section 5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses the limitations of the work in Section 7.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: The theoretical results in Section 4 have complete proofs in Appendix B.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Experimental details, including datasets, model configurations, and hyperparameters, are provided in Appendix C to reproduce the main results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Code is at https://github.com/MhDang/mosaic. The paper provides full experimental details in Appendix C, and all datasets used are publicly available.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Data details and hyperparameters are specified in Section 5 and Appendix C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report error bars for the run-time experiments in Section 5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We report details in Section 5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper discusses both potential positive and negative societal impacts in Section 7.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All used assets are cited.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The details about the new assets are in Appendix C.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.