A Transformer and Remasking Step

Algorithm 2 Diffusion Step

Require: Transformer $\mathcal{N}_n$, full output length n, prompt $\boldsymbol{p}$, input prompt length m, block length d, current diffusion step i, total diffusion steps T, vocabulary V.

1: $\boldsymbol{x} \leftarrow \boldsymbol{p} \cdot \perp^d$ ▷ Pad the input prompt where $n = m + d$
2: $\boldsymbol{v_1} \dots \boldsymbol{v_n} \leftarrow \mathcal{N}_n(\boldsymbol{x})$ ▷ $\boldsymbol{v_i} \in \mathbb{R}_+^{|V|}$ output distribution at position i
3: $\boldsymbol{l} \leftarrow \text{RemaskPositions}(\boldsymbol{v}_{m+1}, \dots, \boldsymbol{v}_{m+d}, i, T)$ ▷ Decides which positions to remask
4: **for** $j \in \boldsymbol{l}$ **do**
5: $\quad \boldsymbol{v_j} \leftarrow \mathbf{0}$
6: $\quad \boldsymbol{v_j}[\perp] \leftarrow 1$ ▷ Set probability of all tokens except $\perp$ to 0.
7: $\boldsymbol{r} \leftarrow D_{m,n}(\boldsymbol{v_1} \dots \boldsymbol{v_n})$ ▷ Decoding that outputs response with first m tokens are input prompt $\boldsymbol{p}$
8: **return r**

We describe the two key components of a single diffusion step: a) **Transformer step:** Computes the output distribution over all tokens in the vocabulary (line 2 Algo. 2). b) **Remasking step:** Based on the output from the transformer step, it greedily decides which token positions to mask. The remasking step can be viewed as updating the output distribution such that, at the masked positions, the mask token $\perp$ is assigned probability 1, while all other tokens receive probability 0 (lines 3 – 6 Algo. 2). Popular greedy remasking strategies include (line 3 Algo. 2): (i) *Random:* Masks tokens at randomly selected positions Nie et al. [2025]. (ii) *Top token probability:* Masks positions where the top-predicted token has the lowest probability Nie et al. [2025]. (iii) *Entropy-based:* Computes the entropy of the output distribution at each position and masks the positions with the highest entropy Ye et al. [2025].

The number of token positions to remask at the i-th step typically depends on the total number of diffusion steps T and the block length d. At step 0, all d positions are masked, and the number of masked tokens decreases linearly to 0 over T steps. Thus, at the i-th step, the number of masked tokens is given by $\left\lfloor \frac{d \times (T-i)}{T} \right\rfloor$.

B Proofs

Proposition 4.1. *[Correctness] Given any regular expression $\mathcal{R}$, input prompt $\boldsymbol{p} \in V^m$, block length d, output distribution $\mathcal{D}_{m+d} = \boldsymbol{v}_1 \dots \boldsymbol{v}_{m+d}$, if $L_P(\mathcal{R}) \cap (V \setminus \perp)^d \neq \{\}$ and $\boldsymbol{r} \sim \boldsymbol{v}_{m+1} \dots \boldsymbol{v}_{m+d}$ be the decoded string, then $\exists \boldsymbol{x} \in V^*.(\boldsymbol{x} \in \mathcal{S}(\boldsymbol{r})) \wedge (\boldsymbol{x} \in L_P(\mathcal{R}))$ holds.*

Proof. We assume that $\exists \boldsymbol{x} \in L_P(\mathcal{R}) \cap (V \setminus \perp)^d \wedge (P(\boldsymbol{x} | \boldsymbol{v}_{m+1} \dots \boldsymbol{v}_{m+d}) \geq 0)$ then the decoded string $\boldsymbol{r}$ satisfy the soundness property (see Definition 3.2). In other words, if there is at least one fully unmasked valid prefix with non-zero probability then DINGO retrieves a valid string.

We show this by induction on the position of tokens. Before moving to the proof, we first define extended transition function δ^* when $\delta : Q \times V \rightarrow 2^Q$ outputs a set of states instead of single state due to mask token $\perp$. In this case, for any string $\boldsymbol{w} \in V^*$, $\delta^*(\boldsymbol{w}, q_0)$ represents the state of reachable states starting from q_0. This can be defined as $\delta^*(\{\}, q_0) = \{q_0\}$ and $\delta^*(t_1 \cdots t_{m+1}, q_0) = \cup_{q \in \delta^*(t_1 \cdots t_m, q_0)} \delta(q, t_{m+1})$.

1. Let $0 \leq i \leq d$, and let $t_1 \dots t_i \in V^i$ denote any token sequence with positive probability mass $\prod_{j=1}^{i} \boldsymbol{v}_{m+j}[t_j] > 0$. Let $q \in \delta^*(t_1 \cdots t_i, q_0)$. Then, $W[i, q] > 0$. We prove this using induction on i.
   (a) Base case $i = 0$: For empty strings only start state q_0 is reachable. DINGO initializes $W[0, q_0] = 1 > 0$ and for all $q \neq q_0$, $W[0, q] = 0$. (lines 1 – 3 in Algo. 1).
   (b) Inductive Step: At position $i + 1$, let $t_1 \dots t_{i+1} \in V^{i+1}$ s.t. $\prod_{j=1}^{i+1} \boldsymbol{v}_{m+j}[t_j] > 0$. Let $q' \in \delta^*(t_1 \cdots t_i, q_0)$ and $q \in \delta(q', t_{i+1})$. By the inductive hypothesis, for all such q'

$W[i, q'] > 0$. Recall,

$$V_{i+1}(q, q') = \begin{cases} \max\limits_{t \in V} \boldsymbol{v}_{m+i+1}(t) \text{ s.t. } q \in \delta(q', t) \\ 0 \text{ if } q, q' \text{ are not connected} \end{cases} \qquad W[i+1, q] = \max_{q' \in Q} W[i, q'] \times V_{i+1}(q, q')$$

Thus, $V_{i+1}(q, q') \geq \boldsymbol{v}_{m+i+1}(t_{i+1}) > 0$ which implies $W[i, q'] \times V_{i+1}(q, q') > 0$. Therefore, $W[i+1, q] = \max_{q' \in Q} W[i, q'] \times V_{i+1}(q, q') > 0$.

2. Since $L_P(\mathcal{R}) \cap (V \setminus \bot)^d \neq \{\}$ by assumption, there exists some $\boldsymbol{y} \in L_P(\mathcal{R}) \cap (V \setminus \bot)^d$. By the Definition 2.6, $q_l = \delta_t^*(\boldsymbol{y}, q_0) \in Q_l$. From the induction above, $W[d, q_l] > 0$. From line 16 in Algo. 1, $q_{max} = \arg\max_{q \in Q_l} W[d, q]$. Thus, by the definition of $\arg\max$, $W[d, q_{max}] \geq W[d, q_l] > 0$.

3. In lines 20-22 in Algo. 1), DINGO reconstructs a d-length sequence $r = t_1 \dots t_d \in V^d$ such that $q_{max} \in \delta^*(r, q_0)$. For any $t_j \in r$, if $t_j = \bot$, choose any token $\tau_j \in (V \setminus \bot)$ satisfying $\delta_t(q_{j-1}, \tau_j) = q_j$ where $q_j = \delta_t^*(t_1 \dots t_j, q_0)$. By definition of $\delta_\bot$, τ_j exists. Substituting every $\bot$ in this manner yields, by Definition 3.1, $\boldsymbol{x} = \boldsymbol{x_1} \dots \boldsymbol{x_d} \in (V \setminus \bot)^d$. $\boldsymbol{x} \in \mathcal{S}(\boldsymbol{r})$. $\delta_t^*(\boldsymbol{x}, q_0) = q_{max}$. From above, $W[d, q_{max}] > 0$.

4. Since $q_{max} \in Q_l$, by Definition 2.6, $\exists w \in \Sigma^*$ s.t. $\delta^*(\boldsymbol{w}, q_{max}) \in F$. Equivalently, $\boldsymbol{x} \cdot w \in L(\mathcal{R})$, hence $\boldsymbol{x} \in L_P(\mathcal{R})$.

□

Proposition 4.2. *[Optimality] Given any regular expression $\mathcal{R}$, input prompt $\boldsymbol{p} \in V^m$, block length d, output distribution $\mathcal{D}_{m+d} = \boldsymbol{v}_1 \dots \boldsymbol{v}_{m+d}$, if $L_P(\mathcal{R}) \cap (V \setminus \bot)^d \neq \{\}$ and $\boldsymbol{r}^* \sim \boldsymbol{v}_{m+1} \dots \boldsymbol{v}_{m+d}$ be the decoded string, then for any valid string $\boldsymbol{r'}$ satisfying $\exists \boldsymbol{x} \in V^*.(\boldsymbol{x} \in \mathcal{S}(\boldsymbol{r'})) \wedge (\boldsymbol{x} \in L_P(\mathcal{R}))$, $P(\boldsymbol{r'} \mid \boldsymbol{v}_{m+1} \dots \boldsymbol{v}_n) \leq P(\boldsymbol{r}^* \mid \boldsymbol{v}_{m+1} \dots \boldsymbol{v}_n)$.*

Proof. 1. First, we show that $P(\boldsymbol{r}^* \mid \boldsymbol{v}_{m+1} \dots \boldsymbol{v}_n) = W[d, q_{max}]$, or equivalently $\prod_{j=1}^{d} \boldsymbol{v}_{m+j}[\boldsymbol{r_j}^*] = W[d, q_{max}]$. Let $\boldsymbol{r}^* = \boldsymbol{r_1}^* \dots \boldsymbol{r_d}^*$ and $0 \leq i \leq d$. We prove by induction on i that if DINGO's backtracking (lines 19 – 23 in Algo 1) has brought us to state $q \in Q$ at position i, then $W[i, q] = \prod_{j=1}^{i} \boldsymbol{v}_{m+j}[\boldsymbol{r_j}^*]$.

   (a) Base case i = 0: $W[0, q_0] = 1 = \prod_{j=1}^{0} \boldsymbol{v}_{m+j}[\boldsymbol{r_j}^*]$.

   (b) Inductive Step: At position i, let $q', \boldsymbol{r_i}^* = Pr[i, q]$ (line 21 in Algo 1). From lines 14 – 15 in Algo 1, $W[i, q] = W[i-1, q'] \times \boldsymbol{v}_{m+i}(\boldsymbol{r_i}^*)$. By the inductive hypothesis, $W[i-1, q'] = \prod_{j=1}^{i-1} \boldsymbol{v}_{m+j}[\boldsymbol{r_j}^*]$. Thus, $W[i, q] = \prod_{j=1}^{i-1} \boldsymbol{v}_{m+j}[\boldsymbol{r_j}^*] \times \boldsymbol{v}_{m+i}(\boldsymbol{r_i}^*) = \prod_{j=1}^{i} \boldsymbol{v}_{m+j}[\boldsymbol{r_j}^*]$.

   Let $q_d \in \delta^*(\boldsymbol{r_1}^* \dots \boldsymbol{r_d}^*, q_0)$. Since $q_d = q_{max}$ (line 19 in Algo 1), $W[d, q_{max}] = \prod_{j=1}^{d} \boldsymbol{v}_{m+j}[\boldsymbol{r_j}^*] = P(\boldsymbol{r}^* \mid \boldsymbol{v}_{m+1} \dots \boldsymbol{v}_n)$.

2. We show that for every valid string $\boldsymbol{r'} = \boldsymbol{r_1}' \dots \boldsymbol{r_d}'$ satisfying $\exists \boldsymbol{x} \in V^*.(\boldsymbol{x} \in \mathcal{S}(\boldsymbol{r'})) \wedge (\boldsymbol{x} \in L_P(\mathcal{R}))$, $\prod_{j=1}^{d} \boldsymbol{v}_{m+j}[\boldsymbol{r_j}'] \leq W[d, q_{max}]$. Let $0 \leq i \leq d$ and $q \in \delta^*(\boldsymbol{r_1}' \dots \boldsymbol{r_i}', q_0)$. We show that $\prod_{j=1}^{i} \boldsymbol{v}_{m+j}[\boldsymbol{r_j}'] \leq W[d, q]$ using induction on i.

   (a) Base case i = 0: $W[0, q_0] = 1 = \prod_{j=1}^{0} \boldsymbol{v}_{m+j}[\boldsymbol{r_j}']$.

   (b) Inductive Step: At position $i+1$, let $q' \in \delta^*(\boldsymbol{r_1}' \cdots \boldsymbol{r_i}', q_0)$ and $q \in \delta(q', \boldsymbol{r_{i+1}}')$. By the inductive hypothesis, $\prod_{j=1}^{i} \boldsymbol{v}_{m+j}[\boldsymbol{r_j}'] \leq W[i, q']$. Recall,

$$V_{i+1}(q, q') = \begin{cases} \max\limits_{t \in V} \boldsymbol{v}_{m+i+1}(t) \text{ s.t. } q \in \delta(q', t) \\ 0 \text{ if } q, q' \text{ are not connected} \end{cases} \qquad W[i+1, q] = \max_{q' \in Q} W[i, q'] \times V_{i+1}(q, q')$$

   Thus, $\boldsymbol{v}_{m+i+1}(\boldsymbol{r_{i+1}}') \leq V_{i+1}(q, q')$. Hence, $\prod_{j=1}^{i+1} \boldsymbol{v}_{m+j}[\boldsymbol{r_j}'] = \prod_{j=1}^{i} \boldsymbol{v}_{m+j}[\boldsymbol{r_j}'] \times \boldsymbol{v}_{m+i+1}(\boldsymbol{r_{i+1}}') \leq W[i, q'] \times V_{i+1}(q, q') \leq W[i+1, q]$.

Let $q_d \in \delta^*(\boldsymbol{r_1}' \dots \boldsymbol{r_d}', q_0)$. Since $\boldsymbol{x} \in V^*.(\boldsymbol{x} \in \mathcal{S}(\boldsymbol{r'})) \wedge (\boldsymbol{x} \in L_P(\mathcal{R}))$, $q_d \in Q_l$. From line 16 in Algo. 1, $q_{max} = \arg\max_{q \in Q_l} W[d, q]$. Thus, by the definition of $\arg\max$, $W[d, q_d] \leq W[d, q_{max}]$. From the inductive hypothesis above, $\prod_{j=1}^{d} \boldsymbol{v}_{m+j}[\boldsymbol{r_j}'] \leq W[d, q_d] \leq W[d, q_{max}]$.

3. Hence, $P(\boldsymbol{r'} \mid \boldsymbol{v}_{m+1} \dots \boldsymbol{v}_n) = \prod_{j=1}^{d} \boldsymbol{v}_{m+j}[\boldsymbol{r_j}'] \leq W[d, q_{max}] = \prod_{j=1}^{d} \boldsymbol{v}_{m+j}[\boldsymbol{r_j}^*] = P(\boldsymbol{r}^* \mid \boldsymbol{v}_{m+1} \dots \boldsymbol{v}_n)$.

□

C Time complexity analysis of parallelized DINGO DP

Algorithm 3 DINGO DP

Require: q_0, block length d, probability vectors $\boldsymbol{v}_1, \dots \boldsymbol{v}_d$ for the current block, Q_l, Q, δ.

1: $W[0, q] \leftarrow 0$ for all $(q \in Q) \wedge (q \neq q_0)$
2: $W[0, q_0] \leftarrow 1$
3: $Pr[0, q] \leftarrow$ (None, None) for all $(q \in Q)$ ▷ Initialization of the DP
4: $V_i \leftarrow \{\}$ for all $i \in \{1, \dots, d\}$ ▷ maximum token probability transtion $(q' \rightarrow q)$ at position i
5: $T_i \leftarrow \{\}$ for all $i \in \{1, \dots, d\}$ ▷ token for the maximum probability transition $(q' \rightarrow q)$
6: **for** $i \in \{1, \dots, d\}$ **do** ▷ The computation along all d can be parallelized
7: # Parallelize for each $\{1, \dots d\}$
8: **for** $(q \in Q)$ **do**
9: **for** $t \in V$ **do**
10: $q' \leftarrow \delta(q, t)$
11: $V_i(q, q'), T_i(q, q') \leftarrow$ MaxTransition$(\boldsymbol{v}_i, t, q, q')$
12: **for** $i \in \{1, \dots, d\}$ **do** ▷ DP computation loop
13: **for** $(q \in Q) \wedge (q' \in Q)$ **do**
14: **if** $W[i, q] < W[i-1, q'] \times V_i(q, q')$ **then**
15: $W[i, q] \leftarrow W[i-1, q'] \times V_i(q, q')$ ▷ Update maximum probability path to q
16: $Pr[i, q] \leftarrow (q', T_i(q, q'))$ ▷ Update the parents accordingly
17: $q_{max} \leftarrow \arg\max_{q \in Q_l} W[d, q]$
18: **if** $W[d, q_{max}] = 0$ **then** ▷ No valid prefixes
19: **return** None, q_{max}
20: $\boldsymbol{r}^* \leftarrow \{\}, q_{curr} \leftarrow q_{max}$
21: **for** $i \in \{d, \dots, 1\}$ **do** ▷ Decoding the optimal string $\boldsymbol{r}^*$
22: $q_{curr}, t \leftarrow Pr[i, q_{curr}]$
23: $\boldsymbol{r}^* \leftarrow \boldsymbol{r}^* \cdot t$
24: **return** reverse$(\boldsymbol{r}^*)$, q_{max}

The parallelism step at line 6 in Algo. 3 can be efficiently implemented using popular frameworks like PyTorch. With parallelism, the computational depth (i.e., the minimum number of sequential steps) reduces to $O(\max(|Q|^2, |Q| \times |V|) + |Q|^2 \times d)$. For regular expressions, where the number of states $|Q|$ is a small constant, the computational depth becomes $O(|V| + d)$, which is linear in both the vocabulary size $|V|$ and the block length d.

D Semi-Autoregressive

In the semi-autoregressive setup, given an input $\boldsymbol{p} \in V^m$, the output $\boldsymbol{o} \in V^{m+d\times k}$ is generated over k blocks, where each block is computed via a call to the single block diffusion model. The output of the i-th diffusion model call is $\boldsymbol{x}_i = \mathcal{L}_{m_i, n_i}(\boldsymbol{x}_{i-1})$, with $\boldsymbol{x}_0 = \boldsymbol{p}$ and the final output $\boldsymbol{o} = \boldsymbol{x}_k$. The input and output lengths for each block are defined as $m_i = m + (i-1) \times d$ and $n_i = m + i \times d$ for all $1 \leq i \leq k$.

Algorithm 4 Semi-Autoregressive diffusion LLM Generation

Require: diffusion LLM $\mathcal{L}$, prompt $\boldsymbol{p}$, answer length n, block length d, diffusion steps T, vocabulary V, number of blocks k.

$\boldsymbol{x} \leftarrow \boldsymbol{p}$ ▷ Initialize $\boldsymbol{x}$ with input prompt $\boldsymbol{p}$
$\boldsymbol{r} \leftarrow \{\}$ ▷ Intialize the output string
for $i \in \{1, \ldots, k\}$ **do**
  $\boldsymbol{x} \cdot \boldsymbol{r_i} \leftarrow \text{Diffusion}(\boldsymbol{x}, m + (i-1) \times d, d, T, V)$ ▷ $\boldsymbol{r_i} \in V^d$ is i-th output block
  $\boldsymbol{r} \leftarrow \boldsymbol{r} \cdot \boldsymbol{r_i}$
  $\boldsymbol{x} \leftarrow \boldsymbol{x} \cdot \boldsymbol{r_i}$ ▷ Compute the input prompt for the next block
Return $\boldsymbol{r}$

Algorithm 5 Semi-Autoregressive Constrained diffusion LLM Generation

Require: diffusion LLM $\mathcal{L}$, prompt $\boldsymbol{p}$, answer length n, block length d, diffusion steps T, vocabulary V, number of blocks k, regular expression $\mathcal{R}$.

$q_0, Q_l, \delta \leftarrow \text{PreProcess}(\mathcal{R})$ ▷ Pre-compute the dfa start state, live states and δ
$\boldsymbol{x} \leftarrow \boldsymbol{p}$ ▷ Initialize $\boldsymbol{x}$ with input prompt $\boldsymbol{p}$
$\boldsymbol{r} \leftarrow \{\}$ ▷ Intialize the output string
$q_{curr} \leftarrow q_0$ ▷ Intialize the current dfa state the response is at
for $i \in \{1, \ldots, k\}$ **do**
  $\boldsymbol{x} \cdot \boldsymbol{r_i}, q_{next} \leftarrow \text{Diffusion}(\boldsymbol{x}, m + (i-1) \times d, d, T, V, Q_l, \delta, q_{curr})$
  if $q_{next} \notin Q_l$ **then**
    return None ▷ No valid completion
  $\boldsymbol{r} \leftarrow \boldsymbol{r} \cdot \boldsymbol{r_i}$
  $\boldsymbol{x} \leftarrow \boldsymbol{x} \cdot \boldsymbol{r_i}$ ▷ Compute the input prompt for the next block
  $q_{curr} \leftarrow q_{next}$ ▷ Update current DFA state for next block
Return $\boldsymbol{r}$

In the semi-autoregressive setting, after each block, we ensure that the output generated so far ends in a live state from Q_l; otherwise, we return the None string (line 7, Algo. 5). Additionally, we maintain a variable q_{curr} to track the current DFA state at the end of each block. This state is then used as the starting state for the dynamic programming step in the constrained generation of the next block.

E Token Transitions Statistics

Table 3: Token Transitions Pre-Computation Statistics

Model Family	$\|V\|$	GSM-Symbolic Time(s)	GSM-Symbolic #States	JSON-Mode Time(s)	JSON-Mode #States
LLaDA-8B	126349	32.09	40	13.22	169.31
Dream-7B	151667	37.01	40	11.87	169.31

In Table 3, we report the precomputation time and the number of states in the DFA for both tasks. For JSON generation, different regular expressions are used for different schemas; therefore, we report the mean precomputation time and mean number of states. The maximum number of states and precomputation times across all questions are 455 and 17.7 (Dream) 21.3 (LLaDA) seconds, respectively.

F GSM-Symbolic

F.1 GSM-Symbolic Prompt

```
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: +, -, /, //, %, *, and **.

You will always respond in the format described below:
Let's think step by step. <reasoning> The final answer is <<symbolic
     expression>>

There are {t} trees in the {g}. {g} workers will plant trees in the {g} today
     . After they are done, there will be {tf} trees. How many trees did the {
     g} workers plant today?

Let's think step by step. Initially, there are {t} trees. After planting,
     there are {tf} trees. The number of trees planted is <<tf - t>>. The
     final answer is <<tf - t>>.

If there are {c} cars in the parking lot and {nc} more cars arrive, how many
     cars are in the parking lot?

Let's think step by step. Initially, there are {c} cars. {nc} more cars
     arrive, so the total becomes <<c + nc>>. The final answer is <<c + nc>>.

{p1} had {ch1} {o1} and {p2} had {ch2} {o1}. If they ate {a} {o1}, how many
     pieces do they have left in total?

Let's think step by step. Initially, {p1} had {ch1} {o1}, and {p2} had {ch2}
     {o1}, making a total of <<ch1 + ch2>>. After eating {a} {o1}, the
     remaining total is <<ch1 + ch2 - a>>. The final answer is <<ch1 + ch2 - a
     >>.

{p1} had {l1} {o1}. {p1} gave {g} {o1} to {p2}. How many {o1} does {p1} have
     left?

Let's think step by step. {p1} started with {l1} {o1}. After giving {g} {o1}
     to {p2}, {p1} has <<l1 - g>> {o1} left. The final answer is <<l1 - g>>.

{question}
```

Listing 1: Prompt template for the GSM-Symbolic task Mirzadeh et al. [2024].

F.2 GSM-Symbolic Regex

```
(?:(?:(?:(?:(?:(?:[ -;=?-~\n]+))*(?:<<(?:(?:\ ))?(?:(?:(?:(?:(?:[a-j])
     |(?:[0-9]{1,3})|\((?:(?:(?:(?:[a-j])|(?:[0-9]{1,3})|\((?:(?:(?:(?:[a-j])
     |(?:[0-9]{1,3})|\((?:(?:(?:(?:[a-j])|(?:[0-9]{1,3})))(?:(?:(?:(?:\ ))
     ?(?:(?:\+|\-|//|/|%|\*|\*\*))(?:(?:\ ))?(?:(?:(?:[a-j])|(?:[0-9]{1,3}))))
     )*)\)))(?:(?:(?:(?:\ ))?(?:(?:\+|\-|//|/|%|\*|\*\*))(?:(?:\ ))?(?:(?:(?:[
     a-j])|(?:[0-9]{1,3})|\((?:(?:(?:(?:[a-j])|(?:[0-9]{1,3})))(?:(?:(?:(?:\ )
     )?(?:(?:\+|\-|//|/|%|\*|\*\*))(?:(?:\ ))?(?:(?:(?:[a-j])|(?:[0-9]{1,3})))
     ))*)\))))*)\)))(?:(?:(?:(?:\ ))?(?:(?:\+|\-|//|/|%|\*|\*\*))(?:(?:\ ))
     ?(?:(?:(?:(?:[a-j])|(?:[0-9]{1,3})|\((?:(?:(?:(?:(?:[a-j])|(?:[0-9]{1,3})
     |\((?:(?:(?:(?:[a-j])|(?:[0-9]{1,3})))(?:(?:(?:(?:\ ))
     ?(?:(?:\+|\-|//|/|%|\*|\*\*))(?:(?:\ ))?(?:(?:(?:[a-j])|(?:[0-9]{1,3}))))
     )*)\)))(?:(?:(?:(?:\ ))?(?:(?:\+|\-|//|/|%|\*|\*\*))(?:(?:\ ))?(?:(?:(?:[
     a-j])|(?:[0-9]{1,3})|\((?:(?:(?:(?:[a-j])|(?:[0-9]{1,3})))(?:(?:(?:(?:\ )
     )?(?:(?:\+|\-|//|/|%|\*|\*\*))(?:(?:\ ))?(?:(?:\+|\-|//|/|%|\*|\*\*))
     (?:(?:\ ))?(?:(?:(?:(?:[a-j])|(?:[0-9]{1,3})|\((?:(?:(?:(?:(?:[a-j])
     |(?:[0-9]{1,3})|\((?:(?:(?:(?:[a-j])|(?:[0-9]{1,3})|\((?:(?:(?:(?:[a-j])
     |(?:[0-9]{1,3})))(?:(?:(?:(?:\ ))?(?:(?:\+|\-|//|/|%|\*|\*\*))(?:(?:\ ))
     ?(?:(?:(?:[a-j])|(?:[0-9]{1,3}))))))*)\)))(?:(?:(?:(?:\ ))
     ?(?:(?:\+|\-|//|/|%|\*|\*\*))(?:(?:\ ))?(?:(?:(?:[a-j])|(?:[0-9]{1,3})
     |\((?:(?:(?:(?:[a-j])|(?:[0-9]{1,3})))(?:(?:(?:(?:\ ))
     ?(?:(?:\+|\-|//|/|%|\*|\*\*))(?:(?:\ ))?(?:(?:(?:[a-j])|(?:[0-9]{1,3}))))
     )*)\))))))*)\)))(?:(?:(?:(?:\ ))?(?:(?:\+|\-|//|/|%|\*|\*\*))(?:(?:\ ))
     ?(?:(?:(?:[a-j])|(?:[0-9]{1,3})|\((?:(?:(?:(?:[a-j])|(?:[0-9]{1,3})
     |\((?:(?:(?:(?:[a-j])|(?:[0-9]{1,3})))(?:(?:(?:(?:\ ))
     ?(?:(?:\+|\-|//|/|%|\*|\*\*))(?:(?:\ ))?(?:(?:(?:[a-j])|(?:[0-9]{1,3}))))
     )*)\)))(?:(?:(?:(?:\ ))?(?:(?:\+|\-|//|/|%|\*|\*\*))(?:(?:\ ))?(?:(?:(?:[
     a-j])|(?:[0-9]{1,3})|\((?:(?:(?:(?:[a-j])|(?:[0-9]{1,3})))(?:(?:(?:(?:\ )
     )?(?:(?:\+|\-|//|/|%|\*|\*\*))(?:(?:\ ))?(?:(?:(?:[a-j])|(?:[0-9]{1,3})))
     ))*)\))))))*)\))))))*)\))))))*))(?:(?:\ ))?>>)))+(?:(?:\.))?)
```

Listing 2: GSM-Symbolic Regex

F.3 GSM-Symbolic Case Studies

Case Study 1:

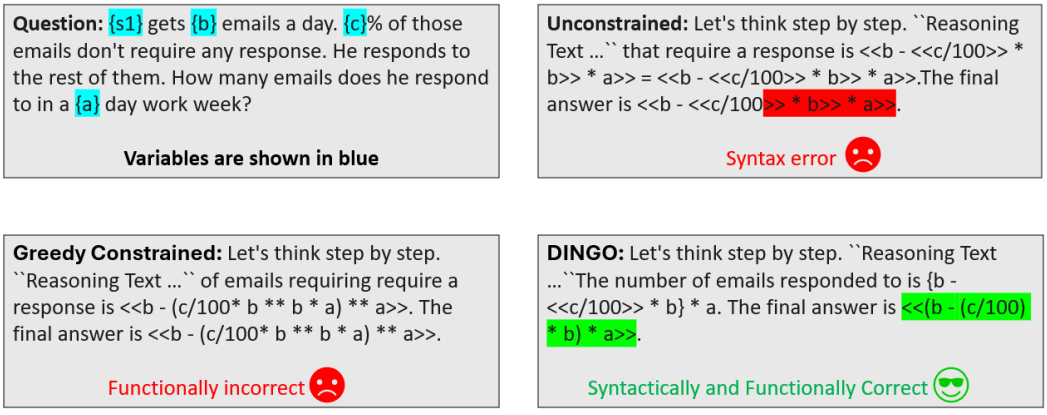

Figure 2: An example from the GSM-symbolic dataset (variables in blue), where unconstrained generation produces syntactically incorrect output, and greedy constrained generation yields a syntactically valid but incorrect answer. In contrast, DINGO generates the correct answer.

Case Study 2:

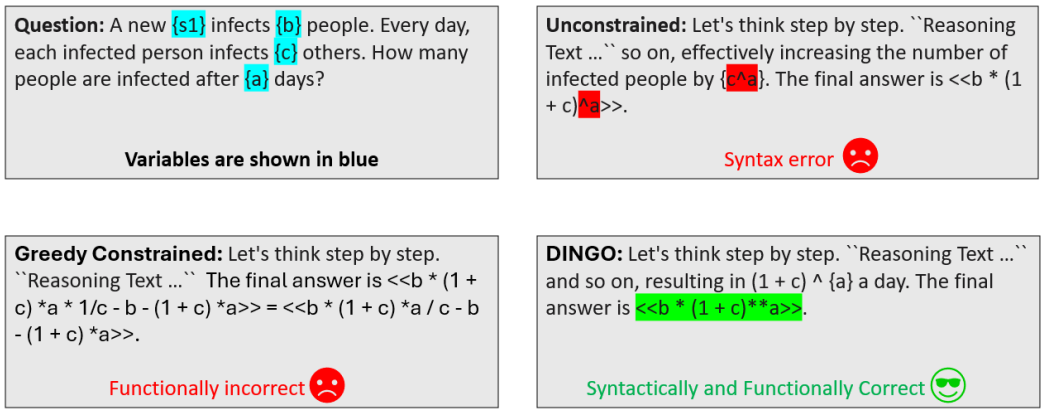

Figure 3: An example from the GSM-symbolic dataset (variables in blue), where unconstrained generation produces syntactically incorrect output, and greedy constrained generation yields a syntactically valid but incorrect answer. In contrast, DINGO generates the correct answer.

G JSON-Mode

G.1 JSON-Mode Example Prompt

```
You are a helpful assistant that answers in JSON. Here's the json schema you
     must adhere to:
<schema>
{'title': 'PromotionalCampaign', 'type': 'object', 'properties': {'campaignID
     ': {'title': 'Campaign ID', 'type': 'string'}, 'productID': {'title': '
     Product ID', 'type': 'string'}, 'startDate': {'title': 'Start Date', '
     type': 'string', 'format': 'date'}, 'endDate': {'title': 'End Date', '
     type': 'string', 'format': 'date'}, 'discountDetails': {'title': '
     Discount Details', 'type': 'string'}}, 'required': ['campaignID', '
     productID', 'startDate', 'endDate']}
</schema>

I'm organizing a promotional campaign for our new eco-friendly laundry
     detergent, which is part of our household products line. The campaign
     will start on June 1, 2023, and end on June 30, 2023. We're planning to
     offer a 15% discount on all purchases during this period. The campaign ID
      is CAMP123456, the product ID is PROD7891011, and the discount details
     are 15% off on all purchases.
Only output the JSON object, no other text or comments.
```

Listing 3: Example JSON Prompt from the JSON-Mode-Eval task NousResearch [2024]. The prompt includes a system message that specifies a schema and a user message that explicitly instructs the model to output a JSON object following that schema with certain parameters.

G.2 JSON-Mode Example Regex

```
\\{[ ]?"campaignID"[ ]?:[ ]?"([^"\\\\\\x00-\\x1F\\x7F-\\x9F]|\\\\["\\\\])*"[
     ]?,[ ]?"productID"[ ]?:[ ]?"([^"\\\\\\x00-\\x1F\\x7F-\\x9F]|\\\\["\\\\])
     *"[ ]?,[ ]?"startDate"[ ]?:[ ]?"(?:\\d{4})-(?:0[1-9]|1[0-2])
     -(?:0[1-9]|[1-2][0-9]|3[0-1])"[ ]?,[ ]?"endDate"[ ]?:[ ]?"(?:\\d{4})
     -(?:0[1-9]|1[0-2])-(?:0[1-9]|[1-2][0-9]|3[0-1])"([ ]?,[ ]?"
     discountDetails"[ ]?:[ ]?"([^"\\\\\\x00-\\x1F\\x7F-\\x9F]|\\\\["\\\\])*")
     ?[ ]?\\}
```

Listing 4: Regex for the JSON Schema in Appendix G.2

G.3 JSON-Mode Case Studies

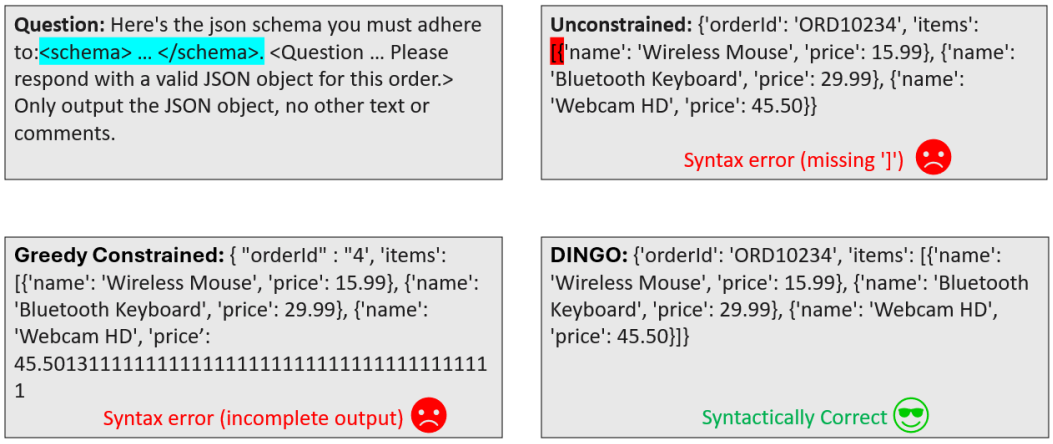

Figure 4: An example from JSON generation, where unconstrained generation produces a syntactically incorrect output, and greedy constrained generation yields a valid but incomplete prefix. In contrast, DINGO generates a syntactically correct answer.

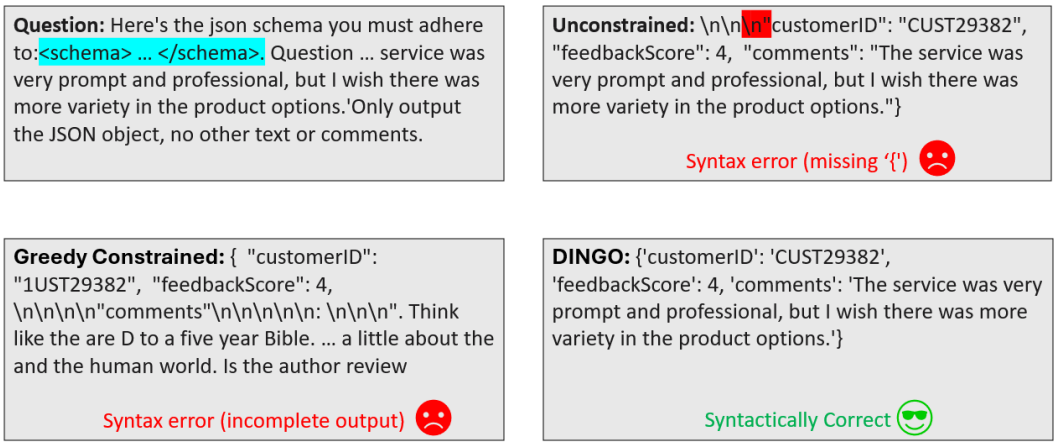

Figure 5: An example from JSON generation, where unconstrained generation produces a syntactically incorrect output, and greedy constrained generation yields a valid but incomplete prefix. In contrast, DINGO generates a syntactically correct answer.

H Ablation Study on Number of Blocks for Diffusion LLM Generation (GSM-Symbolic)

We run generation with a response length of 128, using 64 total diffusion steps, and each of 1, 2, and 8 blocks. Table 4 presents the result.

Table 4: Ablation Study on The Number of Diffusion Blocks for GSM-Symbolic

Model	#Blocks	Method	Acc. (%)	Parse (%)	Time (s)
LLaDA-8B-I	1	Unconstrained	20	54	23.66
		Greedy Constrained	26	94	23.7
		Best of Greedy + Unconstrained	26	94	23.66
		DINGO	29	100	23.73
	2	Unconstrained	22	54	23.63
		Greedy Constrained	30	96	23.81
		Best of Greedy + Unconstrained	30	96	23.65
		DINGO	32	100	23.93
	8	Unconstrained	19	35	23.78
		Greedy Constrained	27	98	23.97
		Best of Greedy + Unconstrained	27	98	23.8
		DINGO	**32**	**100**	23.92
Dream-I-7B	1	Unconstrained	28	69	23.56
		Greedy Constrained	32	90	23.64
		Best of Greedy + Unconstrained	32	90	23.65
		DINGO	34	100	23.67
	2	Unconstrained	30	55	23.62
		Greedy Constrained	33	87	23.71
		Best of Greedy + Unconstrained	33	87	23.62
		DINGO	34	100	23.65
	8	Unconstrained	32	61	23.89
		Greedy Constrained	34	93	24.01
		Best of Greedy + Unconstrained	34	93	23.89
		DINGO	**36**	**100**	23.91

I Ablation Study on Number of Blocks for Diffusion LLM Generation (JSON-Mode)

We run generation with a response length of 128, using 64 total diffusion steps, and each of 1, 2, and 8 blocks. Table 5 presents the result.

Table 5: Ablation Study on The Number of Diffusion Blocks for JSON-Mode.

Model	#Blocks	Method	Acc. (%)	Parse (%)	Time (s)
LLaDA-8B-I	1	Unconstrained	87	91	6.7
		Greedy Constrained	78	79	6.81
		Best of Greedy + Unconstrained	99	99	6.73
		DINGO	100	100	6.78
	2	Unconstrained	84	92	6.72
		Greedy Constrained	92	94	6.83
		Best of Greedy + Unconstrained	99	99	6.73
		DINGO	100	100	6.86
	8	Unconstrained	84	89	6.73
		Greedy Constrained	98	98	6.87
		Best of Greedy + Unconstrained	**100**	**100**	6.75
		DINGO	100	100	6.85
Dream-I-7B	1	Unconstrained	85	87	6.4
		Greedy Constrained	30	30	6.51
		Best of Greedy + Unconstrained	91	93	6.43
		DINGO	**100**	**100**	6.55
	2	Unconstrained	79	82	6.47
		Greedy Constrained	37	39	6.68
		Best of Greedy + Unconstrained	86	88	6.5
		DINGO	100	100	6.63
	8	Unconstrained	70	74	6.44
		Greedy Constrained	52	52	6.65
		Best of Greedy + Unconstrained	86	89	6.46
		DINGO	100	100	6.67

J Ablation Study on Number of Steps for Diffusion LLM Generation (GSM-Symbolic)

We run generation with a response length of 128, 1 block, and each of 16, 32, 64, and 128 total diffusion steps. Table 6 presents the result.

Table 6: Ablation Study on The Number of Diffusion Steps for GSM-Symbolic with Dream-I-7B

#Steps	Method	Acc. (%)	Parse (%)	Time (s)
16	Unconstrained	6	20	5.99
	Greedy Constrained	13	78	6.18
	Best of Greedy + Unconstrained	13	78	5.99
	DINGO	**18**	**100**	6.09
32	Unconstrained	18	48	11.96
	Greedy Constrained	25	87	12.06
	Best of Greedy + Unconstrained	25	87	11.96
	DINGO	**28**	**100**	12.03
64	Unconstrained	28	69	23.56
	Greedy Constrained	32	90	23.64
	Best of Greedy + Unconstrained	32	90	23.65
	DINGO	**34**	**100**	23.67
128	Unconstrained	31	74	47.83
	Greedy Constrained	30	89	47.88
	Best of Greedy + Unconstrained	31	90	47.83
	DINGO	**33**	**100**	47.86

K Ablation Study on Number of Steps for Diffusion LLM Generation (JSON-Mode)

We run generation with a response length of 128, 1 block, and each of 16, 32, 64, and 128 total diffusion steps. Table 7 presents the result.

Table 7: Ablation Study on The Number of Diffusion Steps for JSON-Mode with Dream-I-7B

#Steps	Method	Acc. (%)	Parse (%)	Time (s)
16	Unconstrained	54	59	1.51
	Greedy Constrained	32	32	1.62
	Best of Greedy + Unconstrained	68	71	1.52
	DINGO	**100**	**100**	1.6
32	Unconstrained	67	71	3.23
	Greedy Constrained	35	35	3.35
	Best of Greedy + Unconstrained	78	82	3.24
	DINGO	**100**	**100**	3.31
64	Unconstrained	85	87	6.4
	Greedy Constrained	30	30	6.51
	Best of Greedy + Unconstrained	91	93	6.43
	DINGO	**100**	**100**	6.55
128	Unconstrained	85	87	13.42
	Greedy Constrained	46	46	13.53
	Best of Greedy + Unconstrained	95	97	13.43
	DINGO	**100**	**100**	13.51