

# Automata-Based Steering of Large Language Models for Diverse Structured Generation

Xiaokun Luan<sup>[0000-0002-5878-6486]</sup>, Zeming Wei<sup>[0009-0008-2953-0749]</sup>,  
Yihao Zhang<sup>[0009-0002-0284-1367]</sup>, and Meng Sun<sup>\*</sup><sup>[0000-0001-6550-7396]</sup>

School of Mathematical Sciences, Peking University, Beijing, China

{luanxiaokun, sunm}@pku.edu.cn  
{weizeming, zhangyihao}@stu.pku.edu.cn

**Abstract.** Large language models (LLMs) are increasingly tasked with generating structured outputs. While structured generation methods ensure validity, they often lack output diversity, a critical limitation that we confirm in our preliminary study. We propose a novel method to enhance diversity in automaton-based structured generation. Our approach utilizes automata traversal history to steer LLMs towards novel structural patterns. Evaluations show our method significantly improves structural and content diversity while maintaining comparable generation efficiency. Furthermore, we conduct a case study showcasing the effectiveness of our method in generating diverse test cases for testing open-source libraries.

**Keywords:** Large language models · Structured generation · Diversity

## 1 Introduction

Large language models (LLMs) have demonstrated remarkable capabilities in a wide range of tasks [7, 22, 25, 28], leading to their rapid adoption across various domains. As LLMs are increasingly integrated into complex systems, such as AI agents [9, 24] and automated code generation [3, 14], the demand for reliable and precisely formatted outputs has become critical. These downstream applications often require LLMs to produce outputs that strictly adhere to predefined structures, such as JSON schemas, API call formats, XML documents, or formal specifications, which are essential for ensuring parsability, correctness, and interoperability in complex systems. However, standard LLM generation cannot guarantee consistently structured outputs due to issues like hallucination [10], lack of adherence to instructions [13], and inconsistency across multiple outputs [21].

Structured generation techniques have emerged to ensure that LLM outputs conform to specified structural requirements [15]. Prominent examples include tools like Outlines [26], SGLang [31], and XGrammar [6], which leverage finite automata or pushdown automata to guide the token selection process. Invalid tokens not conforming to the specified grammar are filtered out, ensuring that

---

\* Corresponding author.

the generated outputs adhere to the desired structure. While these approaches have been widely adopted for enforcing output validity, their primary focus has been on correctness. The diversity of the generated structured outputs, however, often remains overlooked.

The importance of diversity is well-recognized in unconstrained LLM generation [5,29], where it fosters creativity and broader coverage of potential responses. This need is equally pertinent in structured generation downstream applications. For instance, in software testing, a diverse set of structured test cases (e.g., API call sequences, configuration files) is crucial for achieving high test coverage and uncovering edge cases. Similarly, AI agents can benefit from diverse structured plans to enhance their adaptability and robustness in complex environments. However, the diversity of the outputs from the current state-of-the-art structured generation methods has not been thoroughly explored. Our preliminary study investigates this question by analyzing outputs from the popular structured generation method Outlines. We observe that the generated samples exhibit limited diversity, tending to repeat common structural patterns. This finding highlights that existing methods, even when paired with sampling strategies like adjusting temperature, may not effectively explore the full space of valid, diverse structured allowed by the constraints. This lack of diversity can hinder the performance of downstream applications, as they may rely on a wide range of structured outputs to function effectively.

To bridge this gap, we propose a novel approach to enhance diversity in regular expression constrained generation. Our core idea is to leverage the history of exploration within the guiding automaton during the generation process. Specifically, we monitor the states and transitions traversed by the LLM. This historical information is then used to adaptively adjust the LLM’s token selection probabilities, encouraging the model to explore less frequented paths within the automaton. To maintain generation quality and prevent unproductive exploration, our method incorporates a penalty mechanism to discourage looping in local states and a dynamic range adjustment factor to ensure appropriate guidance intensity. Our methodology, while targeted at finite automata and regular expressions, is generalizable to pushdown automata and context-free grammars.

Evaluations of our proposed method demonstrate significant improvements in both structural diversity and content diversity compared to baseline method. These gains are achieved while maintaining a substantial portion (approximately 88%) of the baseline generation efficiency. We also conduct ablation studies to confirm the effectiveness of our method under various settings. Furthermore, the case study on generating test cases for testing open-source libraries illustrates that our method can produce diverse test cases that achieve a higher code coverage compared to those generated by the baseline method.

The main contributions of this paper are as follows:

1. A novel method to enhance diversity in automaton-based structured generation, which systematically encourages the LLM to explore less frequented paths within the guiding automaton.

2. A comprehensive evaluation of our method against the baseline method, demonstrating significant improvements in output diversity while maintaining comparable generation efficiency.
3. A case study on generating diverse test cases for testing open-source libraries, showcasing the practical applicability of our method in real-world scenarios.

## 2 Background

In this section, we provide a brief overview of large language model generation and structured generation. Later, we describe the notations and concepts of deterministic finite automata used in our work.

### 2.1 Large Language Model Generation

Large language models generate text by predicting the next token in a sequence following an autoregressive manner. This process starts with a prompt, and a token is sampled from the model’s predicted distribution and appended to the input. The generation continues until an end-of-sequence token `EOS` is produced or a maximum length is reached.

Formally, given a vocabulary  $\mathcal{V}$  and an input sequence  $x = (x_1, x_2, \dots, x_n) \in \mathcal{V}^n$ , the language model  $M$  returns a logit vector  $z$  for the next token, which is then converted into a probability distribution over the vocabulary  $\mathcal{V}$  using the softmax function. The next token  $x_{n+1}$  is sampled from the distribution  $\text{softmax}(z)$ , defined as follows.

$$z = M(x_1, x_2, \dots, x_n), \quad (1)$$

$$x_{n+1} \sim \text{softmax}(z), \text{ where } \text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{|\mathcal{V}|} e^{z_j}}. \quad (2)$$

Sampling temperature  $T$  controls the randomness of the sampling process by scaling the logits before applying softmax, i.e.,  $\text{softmax}(z/T)$ . The higher the temperature, the more uniform the distribution becomes, leading to more diverse outputs. For simplicity, we use  $x_{1:n} := (x_1, x_2, \dots, x_n)$  to denote the sequence of tokens generated so far.

### 2.2 Structured Generation

Structured generation methods impose constraints on the generation process to ensure adherence to the given grammar. At each step, tokens that do not conform to the grammar are masked out, and their logits are set to  $-\infty$ , ensuring that invalid tokens are never sampled. Formally, let  $G$  be a grammar defining a set of valid strings over the vocabulary  $\mathcal{V}$ . Given a sequence  $x_{1:n} \in \mathcal{V}^n$ , the structured generation process can be described as follows.

$$z = M(x_{1:n}), \quad (3)$$

$$z' = \text{mask}(x_{1:n}; G) + z, \quad (4)$$

$$x_{n+1} \sim \text{softmax}(z'), \quad (5)$$

where  $\text{mask}(x_{1:n}; G) \subseteq \mathcal{V}$  is a vector of the same size as  $z$  consisting of zeros for valid tokens and  $-\infty$  for invalid tokens. Valid tokens  $\mathcal{V}_{\text{valid}}(x_{1:n}; G)$  is determined by the grammar  $G$  and the current sequence  $x_{1:n}$  to obtain the mask, i.e.,

$$\text{mask}(x_{1:n}; G)_i = \begin{cases} 0, & \text{if the } i\text{-th token } w_i \in \mathcal{V}_{\text{valid}}(x_{1:n}; G), \\ -\infty, & \text{otherwise.} \end{cases} \quad (6)$$

Therefore, the logits of valid tokens are preserved, while the logits of invalid tokens are set to  $-\infty$  in  $z'$ , ensuring that their probabilities are zero. The grammar  $G$  can be defined in various ways, such as regular expressions and context-free grammars.

To efficiently implement the masking process, finite state automata and pushdown automata can be used to represent the grammar and determine the valid tokens at each step. Specifically, the current state (and the stack) of the automaton is maintained during the generation process. When a new token is sampled, the automaton transitions to a new state (and updates the stack) based on the sampled token. The valid tokens for the next step can then be determined by the current state (and stack) of the automaton. Since the vocabulary is finite, the token-level transition table can be pre-computed to enable efficient masking.

### 2.3 Deterministic Finite Automata

Deterministic finite automata (DFA) are a type of finite state machine that can be used to recognize regular languages. A DFA  $\mathcal{A}$  is defined as a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$ , where  $Q$  is a finite set of states,  $\Sigma$  is the alphabet (set of input symbols),  $\delta : Q \times \Sigma \rightarrow Q$  is the transition function, which maps a state and an input symbol to the next state,  $q_0 \in Q$  is the initial state, and  $F \subseteq Q$  is the set of accepting states. The language defined by the DFA is the set of strings that lead to an accepting state when processed by the automaton, i.e.,  $L(\mathcal{A}) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$ , where  $\delta^*$  is the extended transition function inductively defined as  $\delta^*(q, \epsilon) = q$  and  $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$  with  $\epsilon$  being the empty string.

Given a state  $q$  and a string  $w$ , we denote the sequence of states visited when processing  $w$  in state  $q$  as  $\text{States}(q, w) = (q_1, \dots, q_m)$  with  $q_i = \delta^*(q, w[:i])$ , where  $w[:i]$  is the prefix<sup>1</sup> of  $w$  of length  $i$ . Similarly, we denote the sequence of transitions taken during the processing of  $w$  as  $\text{Transitions}(q, w) = (t_1, t_2, \dots, t_m)$ , where  $t_i = (q_{i-1}, w[i], q_i)$  is the transition from state  $q_{i-1}$  to state  $q_i$  on input symbol  $w[i]$ . We assume that the alphabet  $\Sigma$  consists of all input symbols that make up tokens in the vocabulary  $\mathcal{V}$ , i.e.,  $\mathcal{V} \subseteq \Sigma^*$ . In the following, we will use  $w_i$  to denote some string, e.g., a token in the vocabulary  $\mathcal{V}$ , and  $w_i$  with square brackets  $w_i[j]$  to denote the  $j$ -th symbol in  $w_i$ .

To efficiently guide the LLM generation process, one needs to first pre-compute the subset of valid tokens for each state of the DFA and construct a token transition table. For this purpose, we first introduce the concepts of *live*

---

<sup>1</sup> Square brackets (e.g.,  $w_i[j]$  and  $w[i:j]$ ) are for string indexing (0-based) and slicing.

*states* and *dead states*. A state  $q$  is considered *live* if there exists a path from  $q$  to an accepting state, and it is *dead* if there is no such path. We denote the set of live states as  $Q_{\text{live}}$  and the set of dead states as  $Q_{\text{dead}}$ . The set of valid tokens that can be generated from a given state  $q$  is defined as the subset of the vocabulary that can lead to a live state, i.e.,  $\mathcal{V}_{\text{valid}}(q) = \{w \in \mathcal{V} \mid \delta^*(q, w) \in Q_{\text{live}}\}$ . Based on this, we can construct a token transition table  $T$ , where each entry  $T(q, w)$  contains the next state after processing the token  $w$  from state  $q$ , i.e.,  $T(q, w) = \delta^*(q, w)$ .

### 3 How Diverse is Structured Generation?

#### 3.1 A Preliminary Study

Although structured generation methods guarantee that the generated outputs adhere to the given grammar, not all desirable outputs in the grammar can be generated in practice. This is due to the fact that the sampling process is stochastic and influenced by the model’s prediction, which in turn relies on the training data that may impose biases.

In this section, we conduct a preliminary study to investigate the diversity of structured generation. We select representative, non-trivial regular expressions as constraints, and generate 1000 samples for each grammar using a state-of-the-art structured generation method. To evaluate the diversity of the generated samples, we measure finite automata coverage and count Distinct N-grams. We also briefly examine the effect of varying sampling temperature on output diversity. In the rest of this section, we present the details of our experimental setup and findings.

#### 3.2 Sample Generation

Two regular expressions are selected for our study:  $G_{\text{email}}$  [17] and  $G_{\text{color}}$  [11], which are designed to capture *all* valid email addresses and CSS color codes, respectively. These grammars are not as simple as they may seem, as they also cover many edge cases, such as using IPv4 in email addresses and various color formats in CSS. The minimal DFA for each grammar has 43 and 1309 states, and 1594 and 7495 transitions, respectively. The complete regular expressions are provided in Appendix 7.

We use Qwen2.5-1.5B-Instruct, a 1.5B parameter LLM, to generate 1000 samples for each constraint, following the structured generation method proposed by Willard et al. [26]. This method, implemented in the popular Python library `outlines` [2] (over 11k GitHub stars at the time of writing), has proven to be effective for such tasks. The rationale of this method has been presented in Section 2. We adopt the default multinomial sampling strategy with a temperature of 1.0 and 1.5, representing the normal and high sampling temperatures, respectively. All samples are generated independently with a maximum length of 18 tokens, which is sufficient to cover most valid cases.

Table 1: Diversity evaluation results of preliminary study.

Metric	$G_{\text{email}}$		$G_{\text{color}}$	
	Temp. 1.0	Temp. 1.5	Temp. 1.0	Temp. 1.5
StateCov (%)	18.60	23.26	16.96	31.55
TransCov (%)	20.45	20.83	7.59	12.26
Average length	53.6	68.4	15.3	12.8
Distinct-2	1610	1558	458	635
Distinct-3	10 172	12 526	1295	1991

### 3.3 Diversity Evaluation

We evaluate the diversity of the generated samples from two perspectives: structural diversity and content diversity. Regarding structural diversity, we measure the coverage of DFA states and transitions by the generated samples. Specifically, we compile the regular expression into a minimal DFA and then count the number of unique states and transitions that are visited by these samples. Intuitively, the more diverse the samples are, the more structural patterns they will cover, leading to a higher coverage of states and transitions. On the contrary, a low coverage indicates that some states and transitions are never reached by the given samples, suggesting that the model is not fully exploring the grammar. Formally, given a minimal DFA  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$  and a set of samples  $S = \{s_i\}_{i=1}^n$  generated by the model, we introduce the following metrics to measure the coverage of the DFA states and transitions:

$$\text{StateCov}(S) = \frac{|\bigcup_{i=1}^n \text{States}(q_0, s_i)|}{|Q|}, \quad (7)$$

$$\text{TransCov}(S) = \frac{|\bigcup_{i=1}^n \text{Transitions}(q_0, s_i)|}{|\{(q, a, q') \in Q \times \Sigma \times Q \mid \delta(q, a) = q'\}|}, \quad (8)$$

where  $\text{States}(q_0, s_i)$  and  $\text{Transitions}(q_0, s_i)$  are the sequences of states and transitions traversed by the minimal DFA  $\mathcal{A}$  when processing the sample  $s_i$ .

To evaluate content diversity, we use the commonly used Distinct N-grams metric [12], which counts the number of unique n-grams in the samples. Formally, given a set of samples  $S$ , the Distinct N-grams metric is defined as follows:

$$\text{Distinct-}n(S) = |\{s[i:i+n] \mid s \in S, |s| \geq n, 0 \leq i \leq |s| - n\}|. \quad (9)$$

This metric captures the local sequence variety of the generated samples. The greater the  $\text{Distinct-}n$  value, the more diverse the samples are.

### 3.4 Results and Findings

The main results of our preliminary study are reported in Table 1, where we additionally report average length of the generated samples. Clearly, the finite

automata coverage values for both grammars are relatively low, indicating that the model has not fully explored the DFA even though it has generated 1000 samples. Regarding the Distinct N-grams metric, the Distinct-2 and Distinct-3 values seem satisfactory. But when considering the average length of the generated samples, the Distinct N-grams values are not as impressive as they appear, suggesting that there are many repeated patterns in the generated samples. For example, there are about 53 600 bigrams in the generated email addresses, but only about 3% of them are unique.

Theoretically, increasing the sampling temperature can make the sampling process more random, which helps the model explore more edge cases. However, as shown in Table 1, the DFA coverage of states and transitions have not improved significantly when we increase the sampling temperature to 1.5. Most edge cases in the grammars are still not covered, despite that we have used a relatively high temperature. For example, the regular expression  $G_{\text{email}}$  allows using double quotes in the local part of the email address (e.g., "user@example.com), but none of the generated samples contain this case. Similarly, the regular expression  $G_{\text{color}}$  allows using HSL and HSLA color formats (e.g., `hs1(120, 100%, 50%)`) and the `1ch` and `1ab` functions, which are also absent in the generated samples.

The results of our preliminary study indicate that the state-of-the-art structured generation method is not fully exploring the grammar, leading to a lack of diversity in the generated samples. Increasing the sampling temperature cannot significantly improve the diversity of the generated samples. This is due to the model’s tendency to follow its prediction based on natural language modeling rather than fully considering the grammar constraints. Such a lack of diversity may significantly limit the applicability of structured generation methods in scenarios where diverse outputs are desired.

## 4 Method

### 4.1 Overview

The lack of diversity in structured generation arises from the model’s non-awareness of the legal paths under the grammar constraints, especially those that are rare in the natural language context. To improve the diversity of structured generation, we encourage the model to explore different paths in the automata by adjusting the logits based on history transitions. This is achieved by keeping track of the transitions made by the model during the entire generation process to identify valid paths in the grammar that are rarely explored. In addition, we penalize the tokens that lead to frequently visited states to avoid looping into local optima. The adjustment terms are adaptively scaled based on the range of the logits to avoid over-penalizing or over-rewarding the tokens.

Algorithm 1 summarizes the overall process of our method, where the main difference between our method and the standard structured generation method is that we adjust the logits based on history transitions. The adjustment is achieved by maintaining two counters during the generation process. In the following, we describe the main components of our adjustment terms in detail.

---

**Algorithm 1:** Diversity-enhanced Structured Generation

---

**Input:** The number of samples  $n$ , a minimal DFA  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$  of the given regex, a token transition table  $T$ , a hyperparameter  $\gamma$

**Output:** A set of generated samples  $S = \{s_i\}_{i=1}^n$

$$S = \{\};$$

Initialize global transition counter  $C$  and local state counter  $C_{\text{loc}}$ ;

**for**  $i = 1$  **to**  $n$  **do**

$$\begin{aligned} q &\leftarrow q_0; \\ s_i &\leftarrow \epsilon; \\ \text{Reset local state counter } C_{\text{loc}}; \\ \text{while } \text{True} \text{ do} \\ &\quad z \leftarrow M(\text{prompt} + s_i); \\ &\quad \text{Compute the range}(q, z) \text{ and } \text{adjust}(q); \\ &\quad z'_j = z_j + \text{mask}(q)_j + \gamma \cdot \text{range}(q, z) \text{ adjust}(q)_j \text{ for all } j; \\ &\quad \text{Sample a token } w \text{ from the distribution Softmax}(z'); \\ &\quad \text{if } w \text{ is EOS then} \\ &\quad \quad \text{break}; \\ &\quad s_i \leftarrow s_i + w; \\ &\quad C_{\text{loc}}(q_{j+1}) \leftarrow C_{\text{loc}}(q_{j+1}) + 1 \text{ for each } q_j \in \text{States}(q, w); \\ &\quad q \leftarrow T(q, w); \\ &\quad C(q_j, q_{j+1}) \leftarrow C(q_j, q_{j+1}) + 1 \text{ for each } q_j \in \text{States}(q_0, s_i); \\ &\quad S \leftarrow S \cup \{s_i\}; \end{aligned}$$

**return**  $S$

---

## 4.2 Encouraging Exploration

Suppose a minimal DFA  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$  of the given regular expression and its corresponding token transition table  $T$  are constructed. By design, the minimal DFA ensures that ambiguities and overlapping transitions are resolved. Recall that a token transition  $T(q, w)$  makes several transitions to the next state  $\delta^*(q, w)$  after generating a new token  $w$ , going through a sequence of states  $\text{States}(q, w)$ . If there are two adjacent states  $q_i$  and  $q_{i+1}$  in this sequence and there have never been any transition moving from  $q_i$  to  $q_{i+1}$  during the whole generation process, then we can encourage the model to sample the token  $w$  in state  $q$  so that a new path is explored. Intuitively, the less such a state pair appears in the whole generation process, the more we should reward the model to sample it, thus potentially increasing the structural diversity of the generated samples.

To achieve this, we maintain a global transition counter  $C$  to keep track of the number of times each state bigram (i.e., state pair) has been taken during the whole generation process. Formally,  $C$  is a mapping from a state pair to the number of times it has been taken, i.e.,  $C : Q \times Q \rightarrow \mathbb{N}$ .  $C(q, q')$  is initialized to zero at the beginning of the generation process. This counter is updated after a

valid sample  $s$  is generated<sup>2</sup>, where  $C(q_i, q_{i+1})$  is incremented by one for each state pair  $(q_i, q_{i+1})$  in the sequence  $\text{States}(q_0, s)$ . With such a global counter, we can encourage the model to explore infrequently visited paths by adjusting the logits of the next token. Specifically, assume the current state is  $q$  and the valid tokens are  $w_1, w_2, \dots, w_k \in \mathcal{V}_{\text{valid}}(q)$ . For each token transition  $T(q, w_i)$ , we measure if it is worth exploring by quantifying the least frequently visited state pair in its transition sequence  $\text{States}(q, w_i)$ , defined as follows:

$$E(q, w_i) = \min_{q_j \in \text{States}(q, w_i)} C(q_j, q_{j+1}). \quad (10)$$

Intuitively,  $E(q, w_i)$  represents how many times the least frequently visited state bigram has been taken in the transition sequence of  $w_i$  from state  $q$ . To encourage the model to explore such less frequently visited paths, we add a reward term  $\text{reward}(q)$  to the logits of the next token, defined as follows:

$$\text{reward}(q)_i = \begin{cases} \frac{\log(1 + \sum_{j=1}^k E(q, w_j))}{1 + E(q, w_i)}, & \text{if } w_i \in \mathcal{V}_{\text{valid}}(q), \\ 0, & \text{if } w_i \notin \mathcal{V}_{\text{valid}}(q), \end{cases} \quad (11)$$

where  $k$  is the number of valid tokens in state  $q$ . The reward term  $\text{reward}(q)$  is added to the logits of the next token  $x_{n+1}$ , so that as the model generates more samples, the reward term will favor under-explored paths and thus increasing the diversity of the generated samples.

### 4.3 Avoiding Looping into Local Optima

However, as the global transition counter  $C$  is only updated after a valid sample is generated, it may result in a situation where the model repeatedly predicts a token that leads back to the same state. For example, after generating a few samples, the reward term may favor certain token  $w_i$  in state  $q$  that leads to the same state  $q$  again, thus causing the model to loop into local optima and fail to reach accepting states.

To avoid this, we introduce a local state counter  $C_{\text{loc}} : Q \rightarrow \mathbb{N}$  to keep track of the number of times each state has been visited during the generation process of a single sample. Therefore,  $C_{\text{loc}}$  is similar to  $C$ , except that it is reset to zero at the beginning of each sample generation and updated after a new token is sampled. We similarly introduce  $m(q, w_i)$  to denote the maximum number of times a state has been visited in the state sequence of a valid token  $w_i$  from state  $q$ , defined as follows:

$$m(q, w_i) = \max_{q_j \in \text{States}(q, w_i)} C_{\text{loc}}(q_j). \quad (12)$$

---

<sup>2</sup> Invalid samples are possible due to reaching maximum tokens before generating the **EOS** token.

Based on this, we employ a penalty term  $\text{penalty}(q)$  to adjust the logits of the next token, defined as follows:

$$\text{penalty}(q)_i = \begin{cases} \beta(1 + m(q, w_i)), & \text{if } w_i \in \mathcal{V}_{\text{valid}}(q), \\ 1, & \text{if } w_i \notin \mathcal{V}_{\text{valid}}(q), \end{cases} \quad (13)$$

where  $\beta$  is a hyperparameter that controls the intensity of the penalty. The reward term  $\text{reward}(q)_i$  is divided by the penalty term  $\text{penalty}(q)_i$  before being added to the logits, i.e.,

$$\text{adjust}(q)_i = \frac{\text{reward}(q)_i}{\text{penalty}(q)_i}. \quad (14)$$

Therefore, the penalty term will prevent the model from repeatedly predicting the same token that leads to the same state, even if the reward term favors it.

#### 4.4 Adaptive Scaling

To better adapt the reward and penalty terms to the varying scale of the logits, we introduce an adaptive scaling method to adjust the reward and penalty terms based on the range of the logits and a hyperparameter  $\gamma$ , i.e.,

$$z'_i = z_i + \text{mask}(q)_i + \gamma \cdot \text{range}(q, z) \text{adjust}(q)_i, \quad (15)$$

where  $z_i$  is the original logits of the token  $w_i$ ,  $\text{mask}(q)_i$  is the mask term for the token  $w_i$ , and  $\text{range}(q, z)$  is the range of the logits in state  $q$ , defined as the difference between the maximum and minimum logits of valid tokens in state  $q$ :

$$\text{range}(q, z) = \max_{w_i \in \mathcal{V}_{\text{valid}}(q)} z_i - \min_{w_i \in \mathcal{V}_{\text{valid}}(q)} z_i. \quad (16)$$

This adaptive scaling method allows the model to adjust the reward and penalty terms based on the scale of the logits, thus avoiding over-penalizing or over-rewarding the tokens. Meanwhile, the hyperparameter  $\gamma$  controls the strength of the adjustment, serving as a lever to balance exploration and exploitation. A higher  $\gamma$  value amplifies the reward for less-frequented paths, encouraging the model to generate more diverse outputs. Conversely, a lower  $\gamma$  value leads the model to rely more on its original high-confidence predictions, effectively refining and improving upon the quality of more probable outputs.

### 5 Evaluation

This section evaluates the proposed method for enhancing diversity in structured generation. We compare its performance with a state-of-the-art baseline, Outlines, focusing on diversity and generation efficiency. Ablation studies are conducted to further assess the method’s performance under high-temperature settings and evaluate the contribution of each component. Finally, we present a case study showcasing the practical application of our method in generating diverse test cases for testing open-source libraries. Our implementation is publicly available<sup>3</sup>.

---

<sup>3</sup> <https://github.com/luan-xiaokun/diverse-structured-generation>

## 5.1 Experimental Setup

**Grammar and Language Model** We conduct our experiments on four regular expressions, including  $G_{\text{email}}$ ,  $G_{\text{color}}$ ,  $G_{\text{json}}$ , and  $G_{\text{bomb}}$ . The first two,  $G_{\text{email}}$  and  $G_{\text{color}}$ , were introduced in our preliminary study in Section 3. They represent common real-world formats with intricate standardized rules and many edge cases. The third grammar,  $G_{\text{json}}$ , describes a simple JSON object with three fields (i.e., name, gender, age) and represents a common application scenario in structured generation. Finally,  $G_{\text{bomb}}$  serves as a negative constraint, allowing any string except those containing the word “bomb” (case-insensitive). Such negative constraints are essential for applications like content moderation and safety filtering. The complete regular expressions are provided in Appendix 7.

We use Qwen2.5-1.5B-Instruct, a 1.5B instruction fine-tuned large language model, for structured generation in our experiments. Unless otherwise specified, we adopt a default multinomial sampling strategy with a temperature of 1.0. The max length of the generated samples is set to 18 tokens, except for  $G_{\text{json}}$ , which is set to 54 tokens. This setting is sufficient for the grammars we are using. The prompts used for the generation are as follows:

- $G_{\text{email}}$ : Give me an email address.
- $G_{\text{color}}$ : Give me a CSS color code.
- $G_{\text{json}}$ : Give me a JSON object, which has three fields: name (a string), gender (male or female), age (an integer).
- $G_{\text{bomb}}$ : Tell me how to make a bomb.

**Baseline and Hyperparameters** We compare our method with the state-of-the-art baseline method Outlines. The hyperparameters  $\beta$  and  $\gamma$  of our method are set to 3 and 0.5, respectively. Both methods use the same default multinomial sampling strategy with a temperature of 1.0 unless otherwise specified. For each experiment, we generate 1000 samples per grammar using both our method and the baseline method on a platform with an NVIDIA GeForce RTX 3060Ti GPU and an AMD Ryzen 9 7900X CPU with 32 GB memory.

**Evaluation Metrics** We employ two sets of metrics to evaluate structural and content diversity. For structural diversity, we measure the DFA state coverage StateCov, transition coverage TransCov, and path coverage PathCov of the generated samples, where the path coverage is defined as the ratio of the number of unique state bigrams covered by the generated samples to the total number of state bigrams in the DFA. For content diversity, we use the Distinct N-grams and the Vendi score [8], a commonly used metric for evaluating the diversity of datasets. The Vendi score requires a semi-definite similarity function over the samples, which is a similarity function for strings in our case. We adopt the weighted-degree kernel with shifts kernel [16] as the string similarity function, which considers the number of common substrings between two strings.

Table 2: Structural diversity evaluation results.

Grammar	DFA Size	StateCov (%)		TransCov (%)		PathCov (%)	
		Baseline	Ours	Baseline	Ours	Baseline	Ours
$G_{\text{email}}$	43 / 1594	18.60	<b>95.35</b>	20.45	<b>31.56</b>	13.68	<b>77.78</b>
$G_{\text{color}}$	1309 / 7495	16.96	<b>62.49</b>	7.59	<b>24.94</b>	9.12	<b>42.05</b>
$G_{\text{json}}$	216 / 10192	31.94	<b>56.48</b>	2.04	<b>6.66</b>	12.60	<b>33.11</b>
$G_{\text{bomb}}$	12 / 1213	50.00	<b>83.33</b>	12.12	<b>28.69</b>	27.45	<b>70.59</b>

Table 3: Content diversity evaluation results.

Grammar	Average Length		Distinct-2		Distinct-3		Vendi Score	
	Baseline	Ours	Baseline	Ours	Baseline	Ours	Baseline	Ours
$G_{\text{email}}$	53.6	51.8	1610	<b>1670</b>	10172	<b>10333</b>	702.1	<b>707.7</b>
$G_{\text{color}}$	15.3	11.1	458	<b>679</b>	1295	<b>2039</b>	98.0	<b>169.9</b>
$G_{\text{json}}$	56.8	53.6	354	<b>1639</b>	657	<b>2881</b>	14.7	<b>56.3</b>
$G_{\text{bomb}}$	82.6	79.5	1080	<b>2025</b>	4335	<b>5655</b>	477.0	<b>491.5</b>

## 5.2 Performance Comparison

**Diversity Enhancement** We first evaluate the performance of our method and the baseline method in terms of structural and content diversity. The results are shown in Table 2 and Table 3, respectively, where the DFA size  $n/m$  indicates that the minimal DFA has  $n$  states and  $m$  transitions. We use bold font to highlight the best results between the two methods for each grammar. Our method significantly outperforms the baseline method in terms of both structural and content diversity across all four grammars. The DFA state coverage, transition coverage, path coverage, and the Vendi score of our method are improved by 45%, 12%, 40%, and 90% on average compared to the baseline method. The average length of the generated samples is slightly lower than that of the baseline method, which is expected since our method encourages the model to explore more diverse paths.

As the minimal DFA has a dead state dedicated to handle invalid inputs, achieving 100% state coverage is impossible. Taking this into account, our method has covered all live states in the DFA of  $G_{\text{bomb}}$ . Another figure worth noting is that the transition table of  $G_{\text{json}}$ 's DFA are very dense, with 10192 transitions in total. This is due to the regex of  $G_{\text{json}}$  has a dot-star pattern, which allows any character to appear in the string. Consequently, its transition coverage is relatively low compared to the other three grammars.

**Efficiency Analysis** The proposed method has a higher computational overhead than the baseline method due to the additional logits adjustment step. We analyze the efficiency of our method by measuring the number of tokens generated per second (TPS) and compare it with the baseline method. Table 4 shows the TPS

Table 4: Tokens generated per second of our method and the baseline method.

Grammar	Baseline (TPS)	Ours (TPS)	Percentage
$G_{\text{email}}$	<b>33.44</b>	32.80	98.09 %
$G_{\text{color}}$	<b>34.39</b>	31.49	91.57 %
$G_{\text{json}}$	<b>23.64</b>	18.16	76.82 %
$G_{\text{bomb}}$	<b>25.69</b>	22.60	87.97 %

Table 5: Structural diversity evaluation results under temperature 1.5.

Grammar	StateCov (%)		TransCov (%)		PathCov (%)	
	Baseline	Ours	Baseline	Ours	Baseline	Ours
$G_{\text{email}}$	23.26	<b>90.70</b>	20.83	<b>32.56</b>	17.95	<b>76.92</b>
$G_{\text{color}}$	31.55	<b>61.65</b>	12.26	<b>24.18</b>	16.78	<b>40.68</b>
$G_{\text{json}}$	33.33	<b>56.48</b>	3.87	<b>6.76</b>	14.34	<b>32.04</b>
$G_{\text{bomb}}$	75.00	<b>83.33</b>	30.26	<b>35.94</b>	50.98	<b>74.51</b>

of both methods for each grammar and the percentage of TPS of our method compared to the baseline method. The results show that our method is slower than the baseline method, with an average TPS of 88.8% of the baseline method. This is mainly caused by the computation of the reward term and the penalty term. However, considering the gain in diversity, the trade-off is acceptable.

### 5.3 Ablation Studies

**High-Temperature Generation** To evaluate the performance of our method under high-temperature settings, we re-generate the samples with a temperature of 1.5. The results are shown in Table 5 and Table 6. As expected, the diversity of samples generated by baseline method improves compared with the default temperature setting. However, our method still outperforms the baseline method in terms of both structural and content diversity. Interestingly, all the metrics of our method slightly decrease compared to the default temperature setting. This occurs because our method and temperature scaling represent two distinct, competing forces for promoting diversity. Our method directly manipulates logits to widen their range, while temperature scaling smooths the final probability distribution. When  $T > 1$ , this smoothing effect partially counteracts our explicit logit adjustments, leading to the observed slight decrease in diversity. Therefore, the slight decrease in the diversity of our method compared with the default temperature setting is expected.

We further analyze the perplexity of the generated sampled constrained by  $G_{\text{bomb}}$  to assess the quality of the generated samples, since  $G_{\text{bomb}}$  is the only grammar that allows natural language generation among the four grammars. The perplexity is calculated using another larger model, Phi4-mini-instruct, which is a 4B instruction fine-tuned LLM. Intuitively, a lower perplexity indicates

Table 6: Content diversity evaluation results under temperature 1.5.

Grammar	Average Length		Distinct-2		Distinct-3		Vendi Score	
	Baseline	Ours	Baseline	Ours	Baseline	Ours	Baseline	Ours
$G_{\text{email}}$	68.4	63.6	1558	<b>1814</b>	12 526	<b>12 585</b>	792.0	<b>802.8</b>
$G_{\text{color}}$	12.8	11.3	635	<b>720</b>	1991	<b>2166</b>	143.1	<b>158.4</b>
$G_{\text{json}}$	59.1	61.7	1132	<b>2604</b>	3127	<b>5570</b>	65.4	<b>121.8</b>
$G_{\text{bomb}}$	89.6	85.8	5154	<b>6805</b>	15 375	<b>17 812</b>	819.8	<b>841.5</b>

Table 7: Perplexity of generated samples constrained by  $G_{\text{bomb}}$ .

Temperature	Baseline (PPL)	Ours (PPL)
1.0	54.6	138.0
1.5	55 133.3	81 710.8

that the generated texts are more natural and fluent. The results are shown in Table 7, where the perplexity of the generated samples under temperature 1.5 is significantly higher than that under temperature 1.0 for both methods. Such significant quality drop is not acceptable for natural language generation. As a result, although the baseline method could achieve a higher diversity with a higher temperature, the generated samples are neither of high quality nor diverse. On the other hand, our method achieves a good balance between diversity and quality under the default temperature setting.

**Component Analysis** We conduct an ablation study to analyze the effectiveness of each core component of our method, including the reward term  $\text{reward}(q)$ , the penalty term  $\text{penalty}(q)$ , and the logits range adjustment factor  $\text{range}(q, z)$ . We remove one component at a time and evaluate the performance of the modified method on the  $G_{\text{color}}$  grammar. Table 8 shows the results of the ablation study.

We first remove the reward term by setting  $\text{reward}(q)_i = 1$  for all  $i$ . As expected, the performance drops significantly, even a bit lower than the baseline method. This demonstrates that the reward term is essential for our method to enhance diversity. After removing the penalty term by setting  $\text{penalty}(q)_i = 1$  for all  $i$ , we observe that the generation process becomes unstable and that the model struggles to generate valid samples. As more samples are generated, the model tends to repeat certain patterns (e.g., “oklch(18.27777...)” that quickly reach the max token limit, resulting in a very low success rate on generating valid samples. Therefore, we conclude that the penalty term is crucial for the stability of our method, as it avoids the model from getting stuck in a local optimum. We further remove the logits range adjustment factor. The structural diversity metrics are slightly better than the baseline method, and the content diversity metrics are on par with the baseline method. This indicates that without the adaptive scaling provided by the logits range factor, the strength of the adjustment is insufficient

Table 8: Ablation study on the effectiveness of each component.

Components	DFA Coverage (%)			Distinct-2	Distinct-3	Vendi Score
	StateCov	TransCov	PathCov			
All	62.49	24.94	42.05	679	2039	169.9
No reward( $q$ )	15.30	7.20	8.20	463	1213	93.7
No penalty( $q$ )	—	—	—	—	—	—
No range( $q, z$ )	23.50	9.40	12.70	461	1278	96.8

Table 9: Branch coverage of generated test cases.

Grammar	Library	LoC	Branch Coverage (%)	
			Baseline	Ours
$G_{\text{email}}$	<code>email_validator</code>	792	46.19	<b>59.08</b>
$G_{\text{color}}$	<code>webcolors</code>	441	78.04	<b>83.18</b>

to significantly improve diversity. In summary, all three components contribute to the overall performance of our method.

#### 5.4 Case Study: Test Case Generation

To evaluate the effectiveness of our method in downstream tasks, we conduct a case study on using LLMs to generate test cases for testing open source third-party libraries. We select two popular Python libraries, `email_validator` [23] and `webcolors` [4], which are mainly used for validating email addresses and converting CSS color codes, respectively.

We utilize the generated samples of  $G_{\text{email}}$  and  $G_{\text{color}}$  generated by baseline and our method to test these libraries. The branch coverage of the generated test cases is reported in Table 9. The results show that our method achieves a higher branch coverage than the baseline method with 12.89% and 5.14% improvement on `email_validator` and `webcolors`, demonstrating its effectiveness in generating diverse test cases. As most generated samples are valid, they cannot trigger part of the library’s logic intended to handle invalid inputs. On the other hand, some samples adhere to the given regular expressions but are not valid in the default context of the library, which contributes to the branch coverage. For example, the `email_validator` library does not accept IPv4 addresses in the email field by default<sup>4</sup>, but such variations are valid according to the regex.

To summarize, the proposed diversity-enhanced structured generation method can generate more diverse test cases than the baseline method, which is beneficial for improving the branch coverage of the generated test cases.

<sup>4</sup> Such feature is only supported when a corresponding option is turned on.

## 6 Related Work

### 6.1 Structured Generation of Large Language Models

Structured generation is proposed to improve the quality of LLM-generated outputs by enforcing constraints during the generation process. Various methods have been developed to facilitate structured generation, including Guidance [1], Outlines [26], XGrammar [6], and SGLang [31]. Among them, Guidance employs a template-based approach, where users define templates with placeholders and the model fills in the placeholders with generated tokens. In contrast, Outlines proposes a finite state machine-based approach to mask invalid tokens based on the current automaton state, and it supports both regular expressions and context-free grammars (CFGs). XGrammar focuses on building an engine for efficient CFG-constrained generation through token mask caching and other various optimization techniques. SGLang functions as a domain-specific language embedded in Python, featuring a rich front end for defining structured tasks and a highly optimized back end for efficient execution. Except for Guidance’s template system, these prominent methods are fundamentally based on finite automata or pushdown automata. Their primary distinctions often lie in the specific optimization strategies for their generation and the expressiveness of the primitives they offer to users. Consequently, while these methods differ in usability and generation efficiency, they have generally not prioritized the diversity of the generated structured outputs, often yielding a limited range of structural variations. Our proposed method is orthogonal to these existing engines and front ends, allowing for easy integration into them to enhance their output diversity.

### 6.2 Diversity Enhancement in Text Generation

Diversity is widely recognized as a critical attribute of high-quality text generation. Before the widespread adoption of LLMs, the importance of diversity was already acknowledged, and various methods were proposed to promote it. For instance, Shi et al. [20] employed inverse reinforcement learning to learn a reward function that encourages diverse outputs. Xu et al. [27] proposed Diversity-Promoting Generative Adversarial Networks that assign lower reward for repetitive text and higher reward for novel generations. Following the advent of LLMs, increasing sampling temperature has been a common heuristic to boost output diversity, though often at the expense of coherence or factual accuracy [5]. The trade-off between diversity and other quality aspects is a recurring theme in text generation research. For example, Shao et al. [18] introduced a controllable text generative model based on Conditional Variational Autoencoders to manage this balance via a tunable hyperparameter. Zhao et al. [30] presented a novel method called LoFT to enhance diversity while maintaining faithfulness in logical table-to-text generation. To generate long and diverse text outputs from structured data, Shao et al. [19] proposed a Planning-based Hierarchical Variational Model, which segments input data into a sequence of groups and generates sentences for each group. Our approach is complementary to these methods, specifically targeting the enhancement of diversity in automaton-based structured generation.

## 7 Conclusion and Future Work

In this paper, we identified the limitation of state-of-the-art structured generation methods in producing diverse outputs. To enhance diversity, we proposed a novel method that encourages LLMs to explore new paths in the automata by leveraging the history of traversed states and transitions. Our evaluations demonstrated that our method significantly improves both structural and content diversity while maintaining comparable generation efficiency. We also conducted ablation studies to show that simply increasing sampling temperature in standard structured generation is insufficient for achieving substantial diversity improvements and could degrade output quality. Our case study on generating structured test cases for software testing further illustrated the practical benefits of our approach. Our findings highlight the importance of exploring the full space of valid structured outputs and suggest that our method can be a valuable addition to existing structured generation techniques.

For future work, we plan to generalize our method to more expressive Context-Free Grammars (CFGs) and develop adaptive strategies for the exploration-exploitation trade-off. Furthermore, integrating with parser combinators could enable application to general-purpose programming languages, enhancing tasks like automated code generation.

**Acknowledgments.** This work was supported by the National Key R&D Program of China under Grant 2022YFB2702200, National Natural Science Foundation of China (Grant No. 62172019), and Beijing Natural Science Foundation (Grant Nos. QY24035, QY23041).

## References

1. guidance ai: Guidance: A guidance language for controlling large language models. <https://github.com/guidance-ai/guidance>, accessed: 2025-05-11 16
2. dottxt ai: Structured text generation. <https://github.com/dottxt-ai/outlines>, accessed: 2025-05-11 5
3. Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., Sutton, C.: Program synthesis with large language models (2021). <https://doi.org/10.48550/arXiv.2108.07732>, <https://arxiv.org/abs/2108.07732> 1
4. Bennett, J.: A library for working with HTML/CSS color formats in Python. <https://github.com/ubernostrum/webcolors>, accessed: 2025-05-11 15
5. Chung, J., Kamar, E., Amershi, S.: Increasing diversity while maintaining accuracy: Text data generation with large language models and human interventions. In: Rogers, A., Boyd-Graber, J., Okazaki, N. (eds.) Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 575–593. Association for Computational Linguistics, Toronto, Canada (Jul 2023). <https://doi.org/10.18653/v1/2023.acl-long.34>, <https://aclanthology.org/2023.acl-long.34/> 2, 16
6. Dong, Y., Ruan, C.F., Cai, Y., Lai, R., Xu, Z., Zhao, Y., Chen, T.: XGrammar: Flexible and efficient structured generation engine for large language models (2024), <https://arxiv.org/abs/2411.15100> 1, 16
7. Du, Z., Qian, Y., Liu, X., Ding, M., Qiu, J., Yang, Z., Tang, J.: GLM: General language model pretraining with autoregressive blank infilling. In: Muresan, S., Nakov, P., Villavicencio, A. (eds.) Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 320–335. Association for Computational Linguistics, Dublin, Ireland (May 2022). <https://doi.org/10.18653/v1/2022.acl-long.26>, <https://aclanthology.org/2022.acl-long.26/> 1
8. Friedman, D., Dieng, A.B.: The Vendi score: A diversity evaluation metric for machine learning. Transactions on Machine Learning Research (2023), <https://openreview.net/forum?id=g97OHbQyk1> 11
9. Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N.V., Wiest, O., Zhang, X.: Large language model based multi-agents: A survey of progress and challenges. In: Larson, K. (ed.) Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24. pp. 8048–8057. International Joint Conferences on Artificial Intelligence Organization (8 2024). <https://doi.org/10.24963/ijcai.2024/890>, survey Track 1
10. Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y.J., Madotto, A., Fung, P.: Survey of hallucination in natural language generation. ACM Comput. Surv. **55**(12) (Mar 2023). <https://doi.org/10.1145/3571730> 1
11. Kyza: Pattern matching and extracting color code formats using RegEx. <https://github.com/Kyza/color-regex/>, accessed: 2025-05-11 5
12. Li, J., Galley, M., Brockett, C., Gao, J., Dolan, B.: A diversity-promoting objective function for neural conversation models. In: Knight, K., Nenkova, A., Rambow, O. (eds.) Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. pp. 110–119. Association for Computational Linguistics, San Diego, California (Jun 2016). <https://doi.org/10.18653/v1/N16-1014> 6

13. Lou, R., Zhang, K., Yin, W.: Large language model instruction following: A survey of progresses and challenges. *Computational Linguistics* **50**(3), 1053–1095 (09 2024). [https://doi.org/10.1162/coli\\_a\\_00523](https://doi.org/10.1162/coli_a_00523) 1
14. Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., Savarese, S., Xiong, C.: CodeGen: An open large language model for code with multi-turn program synthesis. In: The Eleventh International Conference on Learning Representations (2023), [https://openreview.net/forum?id=iaYcJKpY2B\\_1](https://openreview.net/forum?id=iaYcJKpY2B_1)
15. OpenAI Platform: Structured model outputs. <https://platform.openai.com/docs/guides/structured-outputs?api-mode=responses> (2024), accessed 12-08-2025 1
16. Rätsch, G., Sonnenburg, S., Schölkopf, B.: RASE: recognition of alternatively spliced exons in *c.elegans*. *Bioinformatics* **21**(1), 369–377 (Jan 2005), <https://doi.org/10.1093/bioinformatics/bti1053> 11
17. Resnick, P.: Internet message format. RFC 5322 (Oct 2008). <https://doi.org/10.17487/RFC5322>, <https://www.rfc-editor.org/info/rfc5322> 5
18. Shao, H., Wang, J., Lin, H., Zhang, X., Zhang, A., Ji, H., Abdelzaher, T.: Controllable and diverse text generation in e-commerce. In: Proceedings of the Web Conference 2021. pp. 2392–2401. WWW ’21, Association for Computing Machinery, New York, NY, USA (2021). <https://doi.org/10.1145/3442381.3449838> 16
19. Shao, Z., Huang, M., Wen, J., Xu, W., Zhu, X.: Long and diverse text generation with planning-based hierarchical variational model. In: Inui, K., Jiang, J., Ng, V., Wan, X. (eds.) Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). pp. 3257–3268. Association for Computational Linguistics, Hong Kong, China (Nov 2019). <https://doi.org/10.18653/v1/D19-1321>, <https://aclanthology.org/D19-1321/> 16
20. Shi, Z., Chen, X., Qiu, X., Huang, X.: Toward diverse text generation with inverse reinforcement learning. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence. pp. 4361–4367. IJCAI’18, AAAI Press (2018) 16
21. Stureborg, R., Alikaniotis, D., Suhara, Y.: Large language models are inconsistent and biased evaluators (2024). <https://doi.org/10.48550/arXiv.2405.01724>, <https://arxiv.org/abs/2405.01724> 1
22. Sun, Y., Wang, S., Feng, S., Ding, S., Pang, C., Shang, J., Liu, J., Chen, X., Zhao, Y., Lu, Y., Liu, W., Wu, Z., Gong, W., Liang, J., Shang, Z., Sun, P., Liu, W., Ouyang, X., Yu, D., Tian, H., Wu, H., Wang, H.: ERNIE 3.0: Large-scale knowledge enhanced pre-training for language understanding and generation (2021), <https://arxiv.org/abs/2107.02137> 1
23. Tauberer, J.: A robust email syntax and deliverability validation library for Python. <https://github.com/JoshData/python-email-validator>, accessed: 2025-05-11 15
24. Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., Anandkumar, A.: Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research* (2024), <https://openreview.net/forum?id=ehfRiF0R3a> 1
25. Wang, H., Xin, H., Zheng, C., Liu, Z., Cao, Q., Huang, Y., Xiong, J., Shi, H., Xie, E., Yin, J., Li, Z., Liang, X.: LEGO-prover: Neural theorem proving with growing libraries. In: The Twelfth International Conference on Learning Representations (2024), <https://openreview.net/forum?id=3f5PAlef5B> 1
26. Willard, B.T., Louf, R.: Efficient guided generation for large language models (2023). <https://doi.org/10.48550/arXiv.2307.09702>, <https://arxiv.org/abs/2307.09702> 1, 5, 16

27. Xu, J., Ren, X., Lin, J., Sun, X.: Diversity-promoting GAN: A cross-entropy based generative adversarial network for diversified text generation. In: Riloff, E., Chiang, D., Hockenmaier, J., Tsujii, J. (eds.) Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. pp. 3940–3949. Association for Computational Linguistics, Brussels, Belgium (Oct-Nov 2018). <https://doi.org/10.18653/v1/D18-1428>, <https://aclanthology.org/D18-1428/> 16
28. Yu, L., Jiang, W., Shi, H., YU, J., Liu, Z., Zhang, Y., Kwok, J., Li, Z., Weller, A., Liu, W.: MetaMath: Bootstrap your own mathematical questions for large language models. In: The Twelfth International Conference on Learning Representations (2024), <https://openreview.net/forum?id=N8N0hgNDrt> 1
29. Zhang, H., Duckworth, D., Ippolito, D., Neelakantan, A.: Trading off diversity and quality in natural language generation. In: Belz, A., Agarwal, S., Graham, Y., Reiter, E., Shimorina, A. (eds.) Proceedings of the Workshop on Human Evaluation of NLP Systems (HumEval). pp. 25–33. Association for Computational Linguistics, Online (Apr 2021), <https://aclanthology.org/2021.humeval-1.3/> 2
30. Zhao, Y., Qi, Z., Nan, L., Flores, L.J., Radev, D.: LoFT: Enhancing faithfulness and diversity for table-to-text generation via logic form control. In: Vlachos, A., Augenstein, I. (eds.) Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics. pp. 554–561. Association for Computational Linguistics, Dubrovnik, Croatia (May 2023). <https://doi.org/10.18653/v1/2023.eacl-main.40>, <https://aclanthology.org/2023.eacl-main.40/> 16
31. Zheng, L., Yin, L., Xie, Z., Sun, C., Huang, J., Yu, C.H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J.E., Barrett, C., Sheng, Y.: SGLang: Efficient execution of structured language model programs (2024). <https://doi.org/10.48550/arXiv.2312.07104>, <https://arxiv.org/abs/2312.07104> 1, 16

## Appendix

The regular expressions  $G_{\text{email}}$ ,  $G_{\text{json}}$ , and  $G_{\text{bomb}}$  used in our study are provided in the following listings.  $G_{\text{color}}$  is exceptionally complex as it covers various color formats. Therefore, we provide a GitHub repository link<sup>5</sup> that contains the complete  $G_{\text{color}}$  expression instead of including it here.

```
(?:[a-zA-Z0-9!#$%^&*+/=?_-`{|}~-]+(?:\.[a-zA-Z0-9!#$%^&*+/=?_-`{|}~-]+)*|\\"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\"[\x01-\x09\x0b\x0c\x0e-\x7f])*\\"@(?:[a-zA-Z0-9](?:[a-zA-Z0-9-]*[a-zA-Z0-9])?\\"+[a-zA-Z0-9](?:[a-zA-Z0-9-]*[a-zA-Z0-9])?\\"|((?:((2(5[0-5]|[0-4][0-9])|1[0-9]|1[1-9]?[0-9]))\.)\{3\})(?:((2(5[0-5]|[0-4][0-9])|1[0-9]|1[1-9]?[0-9]))|[a-zA-Z0-9-]*[a-zA-Z0-9]):(?:[\\x01-\\x08\\x0b\\x0c\\x0e-\\x1f\\x21-\\x5a\\x53-\\x7f]|\\"[\x01-\x09\x0b\x0c\x0e-\x7f])+)\")$
```

Listing 1: The regular expression  $G_{\text{email}}$ .

```
\{\\s*\"name\":\\s*\"(?:.+?)\",\\s*\"gender\":\\s*\"(?:fe)?male\",\\s*\"age\":\\s*\\d+\\s*\\}$
```

Listing 2: The regular expression  $G_{\text{json}}$ .

```
(?:[^bB]![bB][^oO]![bB][oO][^mM]![bB][oO][mM][^bB])+
```

Listing 3: The regular expression  $G_{\text{bomb}}$ .

---

<sup>5</sup> <https://github.com/Kyza/color-regex/>