

SALA: Syntax-Aware Logit Adjustment for Open-Ended Text Generation

Ting-Ruen Wei
Santa Clara University
Santa Clara, CA, USA

Hsin-Tai Wu
DOCOMO Innovations
Sunnyvale, CA, USA

Yi Fang
Santa Clara University
Santa Clara, CA, USA

Abstract

Large language models (LLMs) achieve strong performance in text generation but exhibit pronounced syntactic bias, relying heavily on constructions frequently observed during pre-training while underproducing many patterns common in human-written text. Existing decoding strategies primarily operate at the token level through probability truncation and do not explicitly account for linguistic structure, partly due to the computational cost of evaluating syntactic properties during decoding. We propose SALA, a syntax-aware decoding framework that adjusts token logits to encourage the generation of syntactically rare constructions. SALA employs a lightweight linear probe that predicts part-of-speech information from LLM hidden states and token embeddings, enabling efficient syntax-aware re-weighting of token probabilities with minimal computational overhead. The adjustment magnitude is further modulated by the model’s confidence to preserve fluency and coherence. Experiments across multiple models, decoding strategies, and benchmarks demonstrate that SALA consistently improves generation quality and diversity, and yields gains on reasoning tasks, while incurring minimal latency. Our results highlight the importance of incorporating linguistic structure into decoding and suggest that lightweight, post-hoc interventions can effectively mitigate syntactic bias in LLMs.

1 Introduction

Large language models (LLMs) have achieved impressive results in text generation and are often fine-tuned with reinforcement learning to better align with human preferences. Despite these advances, LLMs exhibit a strong syntactic bias, relying heavily on templates seen during pre-training: 76% of templates in model-generated text appear in the pre-training data, compared to only 35% of templates in human-written text (Shaib et al., 2024). This indicates that LLMs generate only a limited subset of the constructions used by humans, creating a noticeable gap between model-generated and human-written text and reducing generation quality. Because models over-rely on familiar patterns from pre-training while many human syntactic constructions remain underrepresented, LLM outputs fail to capture the full richness and variety of human syntax. Addressing this limitation requires methods that promote the production of underrepresented syntactic patterns.

Language models rely on decoding strategies to generate output tokens. Existing approaches typically perform token truncation, filtering the candidate token set based on the token-level probability distribution, without examining the linguistic properties of individual tokens. This design choice arises from two primary challenges. First, tokens do not always correspond to complete linguistic units, such as words or phrases, making it difficult to reason about syntax or semantics at the token level. Second, evaluating each token for its linguistic properties across the full vocabulary at every decoding step would incur significant computational overhead, which is prohibitive for large-scale generation. As a result, conventional decoding strategies focus primarily on token filtering rather than directly adjusting token probabilities.

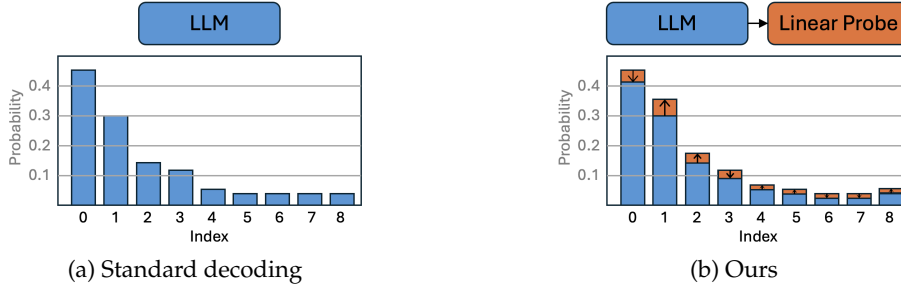


Figure 1: Token distribution before sampling with decoding strategies. Since LLMs rely on syntactic templates seen during pre-training, we design syntax-guided logit adjustment through a linear probe to promote syntactically rare tokens and enhance generation quality.

To alleviate syntactic bias and improve generation quality, we propose SALA, a decoding framework that increases the probabilities of syntactically rare tokens while reducing those of more common tokens, as shown in Figure 1. To address the associated computational overhead, we employ a lightweight linear probe that predicts the part-of-speech (POS) of each token from the LLM’s hidden state and token embedding. Prior work has shown that LLM representations encode substantial syntactic information (Hewitt & Manning, 2019; Tenney et al., 2019), and since both the hidden state and token embedding are readily available during decoding, obtaining these inputs incurs negligible overhead. Token probabilities are then adjusted based on the predicted POS distribution, with the magnitude of adjustment moderated by the model’s confidence to maintain fluency. Our contributions are summarized as follows:

- We propose SALA, a syntax-aware decoding framework that encourages underrepresented syntactic constructions during generation.
- We develop a lightweight linear probe that leverages LLM hidden states to enable efficient logit adjustment with negligible computational overhead.
- We show that SALA improves generation quality and diversity across multiple models and token truncation strategies on open-ended text generation benchmarks, and yields additional gains on math reasoning tasks.

2 Related Work

Decoding Strategies. Existing decoding strategies sample from a filtered set of candidate tokens. Top- k sampling (Fan et al., 2018) draws tokens from the k highest-probability tokens, while top- p sampling (Holtzman et al., 2020) samples from the smallest set of tokens whose cumulative probability mass exceeds p . Mirostat (Basu et al., 2021) dynamically adjusts the effective top- k cutoff to maintain a target perplexity during generation. Typical sampling (Meister et al., 2023) selects tokens whose informativeness is close to the expected entropy of the distribution. Adaptive decoding (Zhu et al., 2024) includes tokens that contribute at least a specified level of confidence. η -sampling (Hewitt et al., 2022) and min- p sampling (Minh et al., 2025) filter out tokens whose probabilities fall below a threshold determined by the entropy and the maximum token probability, respectively. Top- $n\sigma$ sampling (Tang et al., 2025) retains tokens whose probabilities lie within a specified number of standard deviations from the maximum. Shi et al. (2024) and Zhou et al. (2025) investigate in hyperparameter selection among these strategies. However, they operate by filtering or truncating the candidate token set without directly modifying token-level probabilities, which limits their ability to further improve generation quality.

Human-like Text Generation. Research on human-like text generation aims to improve the naturalness, coherence, and overall quality of language model outputs, making them more aligned with human-written text. One prominent approach is reinforcement learning

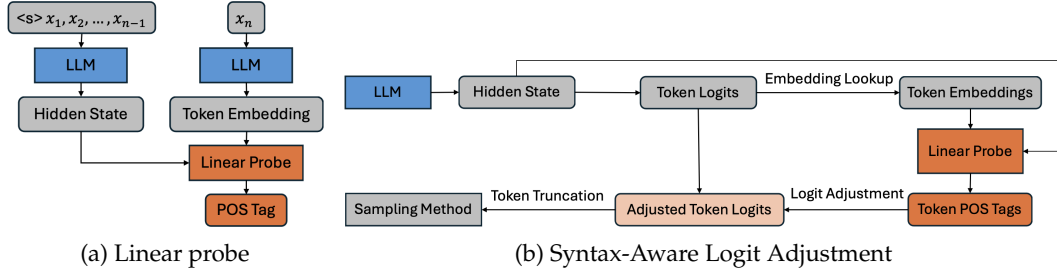


Figure 2: (a) The linear probe is trained to predict the part-of-speech (POS) tag of a token given the previous context, utilizing the hidden state and token embedding for high efficiency. (b) At the decoding stage, the trained linear probe predicts the POS tag of candidate tokens for syntax-aware logit adjustment to encourage syntactically rare tokens and suppress common ones. The adjusted token logits follow any existing sampling method (e.g. top- k) for token truncation and sampling.

from human feedback, where models are fine-tuned using feedback from human evaluators by first training a reward model and then optimizing the language model through reinforcement learning (Stiennon et al., 2020; Ouyang et al., 2022). Other strategies leverage preference-based optimization, such as Direct Preference Optimization (Rafailov et al., 2023), which trains models directly on ranked outputs without requiring an explicit reward model. Additionally, instruction fine-tuning trains models on high-quality human-written instructions and responses to improve generation quality (Wei et al., 2022). These methods improve alignment with human writing by fine-tuning the language model on human judgments or proxies of human preferences, while our approach operates entirely at the decoding stage, requiring no additional fine-tuning of the underlying language model.

3 Methodology

3.1 Problem Formulation

Given an input sequence tokenized by a tokenizer T_a , a language model M_a produces, at each decoding step j , a hidden state $\mathbf{h}_j$ and token logits $\mathbf{L}_j \in \mathbb{R}^{|\mathcal{V}_a|}$, where $\mathcal{V}_a$ denotes the vocabulary of M_a . Our goal is to obtain adjusted logits $\mathbf{L}_j^*$ that yield improved sampling behavior under the same token truncation strategy (e.g., top- k sampling).

3.2 Probing Model

Motivation. To adjust token probabilities during decoding, we require access to the POS information of candidate tokens at each decoding step, while introducing minimal computational overhead. A straightforward approach is to pre-compute a POS tag for each token in the vocabulary and retrieve it during decoding. However, POS tags are inherently context-dependent, making such static annotations error-prone.

Alternatively, POS tags can be obtained using an external POS tagger (e.g., NLTK (Loper & Bird, 2002)). However, this approach has two major limitations: (1) it operates at the word level and often fails to assign meaningful tags to subword tokens, and (2) its sequential nature introduces substantial latency as the sequence length grows.

To address these challenges, we leverage the internal representations of the LLM to predict POS tags via a lightweight linear probe P_a . The hidden states provide contextual information, while the token embedding represents the candidate token itself. Importantly, both are readily available during decoding, making this approach computationally efficient.

Training Setup. We formulate the training data at the token level, where each example is a pair $(\mathbf{x}, \mathbf{y})$, with $\mathbf{x} = (x_1, \dots, x_n)$ and $\mathbf{y} = (y_1, \dots, y_n)$ denoting a token sequence and the corresponding POS tags, respectively. For each position $j \in \{1, \dots, n\}$, we construct a

Algorithm 1 Probing Model Training**Require:** Language model M_a , tokenizer T_a , untrained probe P_a **Ensure:** Trained probe P_a

```

1: for each tokenized sentence  $(\mathbf{x}, \mathbf{y})$  do
2:   for  $j = 1$  to  $|\mathbf{x}|$  do
3:     Run  $M_a$  on prefix  $(S_a, x_{<j})$ 
4:     Extract hidden state  $\mathbf{h}_j$ 
5:     Extract token embedding  $\text{TE}_j$  of  $x_j$ 
6:     Predict POS  $\hat{y}_j \leftarrow P_a(\mathbf{h}_j, \text{TE}_j)$ 
7:     Update  $P_a$  by minimizing  $CE(\hat{y}_j, y_j)$ 
8:   end for
9: end for
10: Return  $P_a$ 

```

training instance by running the language model M_a on the prefix $(S_a, x_{<j})$, where $x_{<j} = (x_1, \dots, x_{j-1})$ and S_a denotes the starting token(s) of M_a . We extract the resulting hidden state h_j , which encodes the contextual information up to position j . The input to the probe P_a is the pair (h_j, TE_j) , where TE_j is the embedding of the target token x_j , and the output is the predicted POS tag of x_j :

$$\hat{y}_j = P_a(h_j, \text{TE}_j) \quad (1)$$

The linear probe concatenates the hidden state and token embedding to predict the POS tag and is trained with cross-entropy loss, as shown in Algorithm 1 and Figure 2a. Since the hidden states and token embeddings are specific to each language model M_a and tokenizer T_a , a separate linear probe P_a must be trained for each model.

Since POS annotations are available at the word level, we adapt a standard word-level POS-tagged corpus (TreeBank (Marcus et al., 1993)) to this token-level formulation, by assigning the same POS tag to all tokens originating from the same word. While this approximation is not always linguistically precise, it provides a consistent and informative syntactic signal that is sufficient for guiding the decoding process. Details can be found in Appendix A.

3.3 Decoding Strategy

With a trained linear probe P_a from Section 3.2, we illustrate the decoding process, as shown in Figure 2. At each decoding index, the LLM produces the hidden state $\mathbf{h}$ and the token logits $\mathbf{L}$. From these logits, we select the top- m candidate tokens and their probabilities $\pi \in \mathbb{R}^m$, retrieve their token embeddings, and predict their POS tags using Equation 1. To promote syntactically rare tokens, we introduce adjustment weights $\mathbf{w} \in \mathbb{R}^m$ based on the predicted POS distribution. For each POS class p , we count the number of tokens assigned to that class among the top- m candidates and compute an entropy-inspired weight:

$$\mathbf{w}_i = \sqrt{-\log_2(c_{p(i)})}, \quad c_p = \frac{|\mathcal{I}_p|}{m} \quad (2)$$

where $\mathcal{I}_p$ is the set of token indices with POS class p , $|\mathcal{I}_p|$ denotes the number of tokens in class p among the top- m candidates, and $p(i)$ denotes the predicted POS class of token i . Consequently, all tokens assigned to the same POS class receive identical weights, with syntactically rarer classes yielding larger adjustments. Intuitively, tokens in less frequent POS classes receive higher weights, thereby increasing the likelihood of sampling syntactically rare tokens.

To balance fluency, we introduce a confidence factor c at each decoding step j to modulate the magnitude of the adjustment. This factor reflects the model’s certainty over the predicted token probabilities: higher uncertainty results in a larger adjustment, while higher

Algorithm 2 SALA**Require:** Language model M_a , tokenizer T_a , trained probe P_a , hyperparameter α , m **Ensure:** Adjusted logits $\mathbf{L}^*$ at each decoding step

```

1: for each decoding step do
2:   Get hidden state  $\mathbf{h}$  and logits  $\mathbf{L}$  from  $M_a$ 
3:   Select top- $m$  tokens and probabilities  $\pi$ 
4:   Retrieve their token embeddings
5:   Predict POS tags for the tokens using  $P_a$ 
6:   for each POS class  $p$  do
7:     Compute class frequency  $c_p = \frac{|\mathcal{I}_p|}{m}$ 
8:   end for
9:   for each candidate token  $i$  do
10:    Compute weight  $\mathbf{w}_i \leftarrow -\log_2(c_{p(i)})$ 
11:   end for
12:   Compute entropy  $c \leftarrow -\sum_m \pi_j \log \pi_j$ 
13:   Normalize entropy  $c \leftarrow c / \log(m)$ 
14:   Adjust logits  $\mathbf{L}^* \leftarrow \mathbf{L} + \alpha \cdot c \cdot \mathbf{w}$ 
15: end for

```

confidence results in a smaller adjustment. We compute c as the normalized entropy of the token distribution:

$$c = -\frac{\sum_{j=1}^m \pi_j \log_2 \pi_j}{\log_2 m} \quad (3)$$

Intuitively, when the model M_a is confident in the next token, the logit adjustment is minor to protect fluency; when the candidate tokens are more evenly likely, the logit adjustment is higher to promote syntactically rare tokens. To control the scale of the adjustment, we normalize the entropy by its maximum possible value, yielding a bounded confidence factor. Finally, we obtain adjusted logits $\mathbf{L}^*$ by:

$$\mathbf{L}^* = \mathbf{L} + \alpha * c * \mathbf{w} \quad (4)$$

where α is a scalar hyperparameter. The algorithm and its GPU-accelerated implementation are shown in Algorithm 2 and Appendix D. The adjusted logits $\mathbf{L}^*$ follow token truncation strategies (e.g. top- k) to sample the output token.

4 Experiments

4.1 Setup

Models. We evaluate our method on several representative LLMs, including GPT2-XL (Radford et al., 2019), Llama2-7B (Touvron et al., 2023), Qwen2.5-7B (Yang et al., 2024), Llama3-8B (Grattafiori et al., 2024), and Llama3.1-70B (Grattafiori et al., 2024). These models span multiple generations and training regimes, enabling evaluation across diverse architectures.

Datasets. We conduct experiments on two widely used text generation benchmarks: WikiText (Merity et al., 2017), which consists of human-written Wikipedia articles, and WritingPrompts (Fan et al., 2018), a story generation dataset containing human-written prompts and narratives.

Baselines. We compare existing decoding strategies in their standard form against the same strategies incorporated with our approach. The strategies are top- k sampling (Fan

	WritingPrompts					WikiText				
	R2	R3	R4	Div.	Mauve	R2	R3	R4	Div.	Mauve
Human	4.27	1.16	0.59	94.37	–	4.24	1.14	0.57	94.37	–
Top- k	21.86	14.94	11.80	65.67	14.67	18.57	13.75	11.62	70.55	13.58
+ SALA	17.88*	11.35*	8.73*	71.75*	14.91*	15.40*	10.68*	8.82*	75.64*	14.40*
Top- p	8.04	4.67	3.57	87.08	18.63	8.87	5.79	4.69	85.73	18.72
+ SALA	7.59*	4.33*	3.28*	87.89	20.30	8.79	5.69	4.59	85.94	19.03
Typical	9.32	5.50	4.16	85.10	18.38	7.02	4.46	3.55	88.58	20.10
+ SALA	6.32*	3.46*	2.63*	89.93*	20.23	6.81*	4.23*	3.35*	88.88	19.70
Min- p	58.53	53.07	49.75	22.14	3.80	38.88	34.80	32.59	43.65	4.15
+ SALA	48.01*	41.62*	38.09*	33.23*	6.60*	33.72*	29.01*	26.67*	50.14*	6.19*
η	10.53	6.29	4.74	83.31	17.20	14.03	9.88	8.19	77.94	17.09
+ SALA	9.18*	5.22*	3.96*	85.36*	18.34	13.53*	9.52*	8.00	78.61	17.33*
Adaptive	17.19	11.43	9.02	73.11	15.96	68.76	67.07	65.91	19.08*	1.12
+ SALA	12.18*	7.39*	5.72*	80.68*	17.99*	70.08	68.36	67.25	17.49	1.17*
Top- $n\sigma$	46.22	39.92	36.35	35.90	7.80	34.98	30.52	28.22	48.52	6.23
+ SALA	35.61*	29.11*	25.81*	48.52*	9.95*	29.53*	24.84*	22.65*	55.76*	8.41*
Human	8.50	2.15	0.76	89.03	–	5.88	1.45	0.53	89.03	–
Top- k	13.18	6.46	4.45	79.90	60.85	9.70	5.01	3.72	85.43	33.80
+ SALA	12.45*	5.87*	4.01*	81.16*	68.37*	9.37*	4.67*	3.44*	86.12*	34.74
Top- p	5.34	2.05	1.43	92.22	64.81	6.11	2.74	1.95	90.99	36.38
+ SALA	4.78*	1.86*	1.36	93.02*	65.14	5.81*	2.51*	1.77*	91.47	37.26
Typical	4.30	1.64	1.20	93.67	60.98	6.15	2.79	2.03	90.97	35.14
+ SALA	4.19	1.55	1.12	93.91	67.64*	4.86*	2.10*	1.53*	92.85	37.30
Min- p	25.48	17.13	13.61	60.40	45.33	20.06	14.12	11.90	68.97	21.57
+ SALA	22.49*	14.52*	11.40*	65.02*	45.90*	18.32*	12.59*	10.57*	71.94*	25.50*
η	8.33	3.62	2.57	87.57	59.17	7.99	3.85	2.80	88.18	32.83
+ SALA	7.80*	3.29*	2.28*	88.40*	63.41*	8.33	4.17	3.13	87.68	34.50*
Adaptive	9.55	4.31	3.04	85.63	63.69	54.44	49.87	47.14	24.38	5.77
+ SALA	9.00*	4.00*	2.80*	86.45	65.34	51.35*	46.60*	44.01*	27.28*	5.86*
Top- $n\sigma$	17.91	10.62	8.04	72.09	51.08	17.30	11.64	9.66	73.67	24.24
+ SALA	15.83*	8.92*	6.70*	75.59*	57.05*	14.84*	9.45*	7.72*	77.66*	26.84*

Table 1: Performance of different decoding strategies across two datasets for GPT2-XL (**top**) and Llama3-8B (**bottom**). Each sample is generated five times and averaged. Bold indicates better performance, while * denotes statistical significance ($p < 0.05$) compared to the baseline. SALA consistently improves the generation quality and diversity.

et al., 2018), top- p sampling (Holtzman et al., 2020), typical sampling (Meister et al., 2023), adaptive decoding (Zhu et al., 2024), η -sampling (Hewitt et al., 2022), min- p sampling (Minh et al., 2025), and top- $n\sigma$ sampling (Tang et al., 2025). Using identical hyperparameters in the decoding strategy, this setup directly measures the effect of our approach on top of each decoding strategy.

Evaluation Metrics. We assess generation quality using Mauve (Pillutla et al., 2021), which measures the similarity between the token distribution of generated text and that of human-written text. We also evaluate diversity using repetition rates of 2-, 3-, and 4-grams, denoted as R2, R3, and R4, respectively (Zhu et al., 2024), where lower values indicate higher diversity. In addition, we report an overall Diversity score (Meister et al., 2023), computed from these repetition metrics. Higher MAUVE and Diversity scores indicate better generation quality and diversity, respectively.

Implementation Details. Following Zhu et al. (2024), we frame the experiments as a text continuation task on 1,200 randomly selected samples from each dataset, with 1,000 samples for testing and 200 for validation. We use the pre-trained base model without further fine-tuning. For WikiText, the first 32 tokens are used as the prompt, while for the story generation task, the provided prefix serves as the prompt. The maximum generation length is 256 tokens. To account for the stochasticity of sampling-based decoding, each

	Qwen2.5-7B-Instruct		Llama3-8B-Instruct		Qwen3-8B	
Method	MAUVE	Div.	MAUVE	Div.	MAUVE	Div.
Human	–	94.24	–	94.40	–	94.24
Top- k	37.59	96.51	38.09	95.76	31.54	94.71
+ SALA	38.11	96.56	38.44	95.74	32.77*	94.97
Top- p	36.12	96.49	36.12	95.60	30.68	94.54
+ SALA	37.76	96.44	36.77	95.48	32.38*	94.53
Typical	33.45	96.73	33.45	95.80	32.87	94.77
+ SALA	37.24	96.81	33.93	95.82	35.45*	95.14
Min- p	37.82	95.62	37.82	94.93	33.38	94.25
+ SALA	39.03*	95.67	38.37	94.97	34.28*	94.22
η -sampling	36.75	96.69	36.75	95.75	32.41	94.72
+ SALA	39.73*	96.72	34.80	95.85	34.07*	95.13
Adaptive	38.69	94.16	30.40	94.18	36.00	93.43
+ SALA	39.78	93.83	30.40	94.18	36.50	93.41
Top- $n\sigma$	34.74	95.95	35.60	94.95	32.53	94.57
+ SALA	37.55*	95.91	35.66	94.92	32.77	94.68

Table 2: Instruction-following performance under Qwen2.5-7B, Llama3-8B, and Qwen3-8B models. SALA consistently improves generation quality across truncation strategies. Div. denotes Diversity, while * indicates statistical significance ($p < 0.05$).

sample is generated five times under a temperature of 1.0, and the results are averaged. The hyperparameter values of the decoding strategies are selected based on previous literature (Zhu et al., 2024; Zhou et al., 2025; Shi et al., 2024) and can be found in Appendix B. All token truncation strategies are implemented using the official implementations in the Hugging Face Transformers library¹.

4.2 Results

Table 1 demonstrates SALA’s performance on two datasets with GPT2-XL and Llama3-8B across seven baseline strategies, evaluating both generation quality and diversity. Overall, SALA improves the diversity and quality of the underlying truncation strategies in most settings, with statistically significant gains in many cases. The largest improvement is observed for min- p sampling on GPT2-XL, with gains of 74% and 50% on the two datasets, respectively. For common truncation strategies such as top- k and top- p sampling, SALA consistently yields improvements, with statistical significance in some cases. In a few settings (e.g., η -sampling with Llama3-8B on WikiText), SALA slightly reduces token diversity, but the generated text still exhibits higher overall quality than the corresponding baseline.

This improvement can be attributed to SALA’s ability to promote syntactically rare tokens and encourage more human-like generation. By encouraging these syntactically rare tokens, SALA enhances generation quality while often increasing diversity, as observed in most settings. In cases where generation quality improves without a corresponding increase in diversity, we hypothesize that SALA promotes tokens that are more aligned with human-like text, even if their overall diversity does not surpass that of the baseline outputs. Similar trends are observed for Llama2-7B. For Qwen2.5-7B and Llama3.1-70B, however, baseline truncation strategies yield very low generation quality (around 5% Mauve), making SALA less stable in these settings. Additional results are provided in Appendix E.

4.3 SALA for Instruction-following

Instead of text continuation, we evaluate SALA on instruction-following tasks using Qwen2.5-7B-Instruct (Yang et al., 2024) and Qwen3-8B (Yang et al., 2025) on the WritingPrompts dataset. The instruction prompt is provided in Appendix C.1. As shown in Table 2, SALA consistently improves generation quality across different truncation strategies in this setting.

¹https://github.com/huggingface/transformers/blob/main/src/transformers/generation/logits_process.py

Decoding	Turn-1	Turn-2	Mean
Top- p	7.5625	6.8875	7.2250
+ SALA	7.7500	7.1250	7.4375

Table 3: Llama3-8B-Instruct performance on the MT-Bench dataset with GPT-4.1 as the scorer.

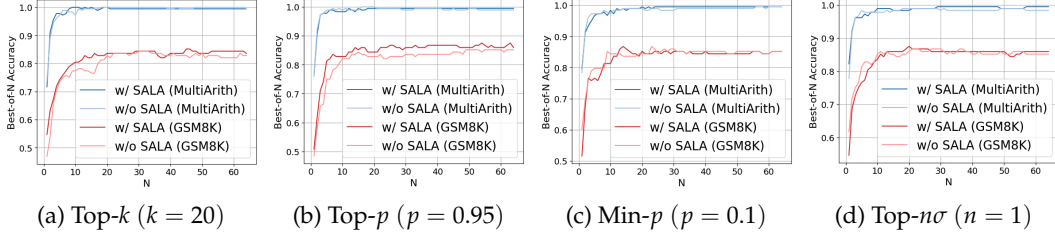


Figure 3: Accuracy vs. number of samples N with and without SALA. SALA improves min- p and top- $n\sigma$ on MultiArith, and consistently boosts top- k and top- p on GSM8K; gains for other methods converge as N increases.

We further evaluate SALA on the real-world MT-Bench dataset (Zheng et al., 2023), which measures instruction-following performance across two conversational turns. For each turn, GPT-4.1 scores the outputs of Llama3-8B-Instruct under top- p truncation, both with and without SALA. As reported in Table 3, SALA consistently improves instruction-following performance across turns.

4.4 SALA for Reasoning

We evaluate SALA on two mathematical question-answering datasets: MultiArith (Roy & Roth, 2015) and GSM8K (Cobbe et al., 2021), using the self-consistency strategy (Wang et al., 2023) with Llama3.2-3B-Instruct (Grattafiori et al., 2024). For each question, the final prediction is determined by majority voting over multiple sampled solutions. The prompt can be found in Appendix C.2. Figure 3 reports accuracy as a function of the number of samples N , comparing each truncation strategy (top- k , top- p , min- p , and top- $n\sigma$) with and without SALA.

On MultiArith, SALA improves performance for both min- p and top- $n\sigma$ sampling, although accuracy for all methods exceeds 95% after a small number of samples. On GSM8K, SALA yields substantial and consistent gains for top- p across most values of N , and improves top- k when the sample size is around 15 and 55. For min- p and top- $n\sigma$, SALA may slightly underperform at small sample sizes, but converges to comparable performance as N increases.

5 Analysis

5.1 Ablation

Effect of m . SALA adjusts the logits of the top- m tokens, leaving the remaining tokens untouched. For small m , tokens with substantial probabilities are excluded. As m increases, the additional tokens carry progressively smaller probabilities, while potentially increasing computational cost. To analyze this trade-off, we examine the effect of varying m on computation time, generation quality, and token probability mass coverage, as shown in Figure 4.

For GPT2-XL, the top-500 tokens (orange) cover 90% of the probability mass in only 20% of cases, indicating a relatively flat output distribution. In contrast, Qwen3-8B exhibits more confident predictions, with the top-300 tokens consistently covering over 99% of the probability mass. Figure 4 further shows that increasing m generally improves generation

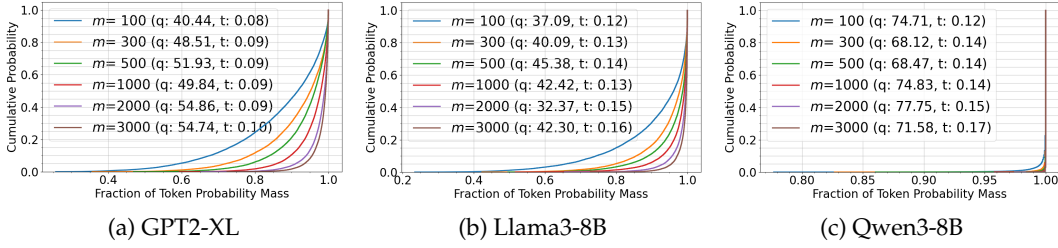


Figure 4: Cumulative distribution of the fraction of probability mass captured by the top- m tokens over multiple iterations for three models and their respective generation quality (q; Mauve) and time cost (t; milliseconds).

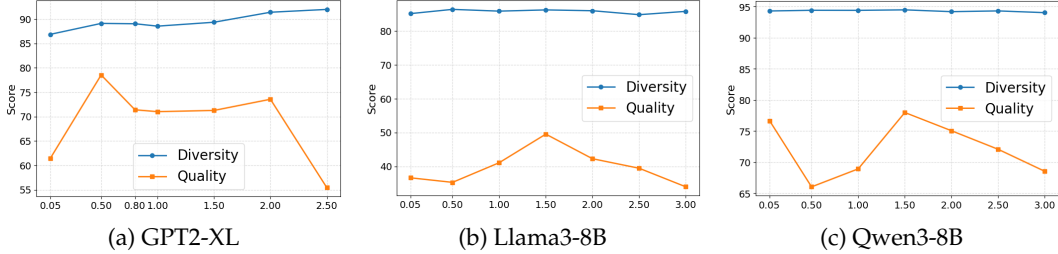


Figure 5: Quality and diversity across hyperparameter α on the WritingPrompts dataset. Quality is sensitive to α and varies across models, with optimal values differing by model, while diversity remains largely stable.

quality when the probability mass is distributed across many tokens. Once most of the probability mass is covered, further increasing m yields diminishing or even negative returns in generation quality. In terms of computational overhead, the additional cost is negligible: expanding m by 1,000 tokens increases runtime by 0.01 ms on average.

Effect of α . The hyperparameter α controls the magnitude of the logit adjustment in SALA: $\alpha = 0$ corresponds to standard decoding, while larger values increase the influence of the syntactic signal. We analyze the effect of varying α on generation quality and diversity, as shown in Figure 5. While diversity remains relatively stable across different values of α , generation quality is more sensitive to this parameter and exhibits model-specific optimal ranges. For GPT2-XL, generation quality improves when $\alpha \in [0.5, 2.0]$, but degrades at higher values due to excessive syntactic influence. For Qwen3-8B, some values of α reduce generation quality, whereas $\alpha = 1.5$ yields the best performance, indicating a model-dependent sweet spot.

5.2 Decoding Speed

Table 4 compares the total decoding latency per step of SALA with existing POS tagging tools, including spaCy² and NLTK³, under the top- p truncation strategy on an NVIDIA H200 GPU. We report the average decoding time per step, averaged over 100 samples. Due to their high computational overhead, spaCy and NLTK are evaluated by obtaining POS tags for only 10 candidate tokens per step, whereas SALA obtains for 1,000 tokens per step.

On average, top- p decoding alone requires less than 10 ms per step, while spaCy and NLTK increase the total latency to approximately 50 ms and 800 ms per step, respectively, making logit adjustment impractical at inference time. In contrast, SALA requires only 0.9-1.4 ms

²<https://spacy.io/usage/linguistic-features#pos-tagging>

³https://www.nltk.org/api/nltk.tag.pos_tag.html

LLM	Top- p	spaCy	NLTK	SALA
GPT2-XL	0.53	51.03	819.11	0.88
Llama3-8B	0.58	51.61	840.01	1.33
Qwen3-8B	0.88	73.18	867.62	1.42

Table 4: Average decoding time (ms) with POS tagging compared against spaCy and NLTK libraries.

	Top- k	Top- p	Min- p	Top- $n\sigma$
	Baseline / +SALA	Baseline / +SALA	Baseline / +SALA	Baseline / +SALA
WikiText	4.35 / 4.36	4.38 / 4.37	4.31 / 4.41	4.31 / 4.42
WritingPrompts	4.36 / 4.44	4.47 / 4.47	4.28 / 4.43	4.29 / 4.42

Table 5: Entropy of syntactic template distributions for Llama3-8B across decoding strategies and datasets. Results are reported with and without SALA; higher entropy is shown in bold.

per decoding step in total. By directly leveraging LLM hidden states, SALA substantially reduces decoding latency compared to external POS taggers such as spaCy and NLTK.

5.3 Syntactic Diversity

We further analyze the syntactic diversity of the generated outputs using dependency-based syntactic templates. Specifically, for each generated sample, we extract templates of the form ((head POS, dependency relation, child POS)). We aggregate these templates across the test set and compute their Shannon entropy over the resulting template distribution. Higher entropy indicates a more uniform distribution of syntactic patterns, reflecting greater structural diversity and less reliance on repetitive dependency structures. As shown in Table 5, SALA generally increases the entropy across decoding strategies and datasets, indicating that SALA promotes greater diversity in the syntactic structures of generated text.

6 Conclusions and Future Work

In this paper, we proposed SALA, a syntax-aware decoding framework that adjusts token logits to encourage syntactically diverse and less frequent constructions during generation. By incorporating syntactic signals directly at the decoding stage, SALA complements existing decoding strategies without requiring any modification or fine-tuning of the underlying model. Experimental results across multiple benchmarks and model families show that syntax-aware logit adjustment consistently improves both generation quality and diversity. The results further indicate improvements on tasks such as math reasoning. By leveraging model hidden states and token embeddings, SALA achieves these gains with substantially lower computational cost compared to existing part-of-speech taggers.

These findings underscore the importance of integrating linguistic structure into the decoding process, demonstrating that lightweight, post-hoc interventions can meaningfully influence model outputs. While SALA is effective across a range of tasks, future work could explore extensions to other languages, broader domains, and additional linguistic signals. More broadly, we hope this work motivates further research to enhance the controllability, diversity, and overall quality of large language model outputs.

References

Sourya Basu, Govardana Sachitanandam Ramachandran, Nitish Shirish Keskar, and Lav R Varshney. Mirostat: A neural text decoding algorithm that directly controls perplexity. In *International Conference on Learning Representations*, 2021.

- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 889–898, 2018.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- John Hewitt and Christopher D Manning. A structural probe for finding syntax in word representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4129–4138, 2019.
- John Hewitt, Christopher D Manning, and Percy Liang. Truncation sampling as language model desmoothing. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 3414–3427, 2022.
- Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020.
- Edward Loper and Steven Bird. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, pp. 63–70, 2002.
- Mitch Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.
- Clara Meister, Tiago Pimentel, Gian Wiher, and Ryan Cotterell. Locally typical sampling. *Transactions of the Association for Computational Linguistics*, 11:102–121, 2023.
- Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.
- Nguyen Nhat Minh, Andrew Baker, Clement Neo, Allen G Roush, Andreas Kirsch, and Ravid Shwartz-Ziv. Turning up the heat: Min-p sampling for creative and coherent llm outputs. In *The Thirteenth International Conference on Learning Representations*, 2025.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.
- Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaid Harchaoui. Mauve: Measuring the gap between neural text and human text using divergence frontiers. *Advances in Neural Information Processing Systems*, 34: 4816–4828, 2021.
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.
- Subhro Roy and Dan Roth. Solving general arithmetic word problems. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pp. 1743–1752, 2015.
- Chantal Shaib, Yanai Elazar, Junyi Jessy Li, and Byron C Wallace. Detection and measurement of syntactic templates in generated text. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 6416–6431, 2024.

- Chufan Shi, Haoran Yang, Deng Cai, Zhisong Zhang, Yifan Wang, Yujiu Yang, and Wai Lam. A thorough examination of decoding methods in the era of llms. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8601–8629, 2024.
- Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in neural information processing systems*, 33:3008–3021, 2020.
- Chenxia Tang, Jianchun Liu, Hongli Xu, and Liusheng Huang. Top- $n\sigma$: Eliminating noise in logit space for robust token sampling of llm. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10758–10774, 2025.
- Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4593–4601, 2019.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023.
- Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022.
- An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.
- Yuxuan Zhou, Margret Keuper, and Mario Fritz. Balancing diversity and risk in llm sampling: How to select your method and parameter for open-ended text generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 26352–26365, 2025.
- Wenhong Zhu, Hongkun Hao, Zhiwei He, Yiming Ai, and Rui Wang. Improving open-ended text generation via adaptive decoding. In *International Conference on Machine Learning*, pp. 62386–62404. PMLR, 2024.

A Part-of-Speech Classification

POS classes. We adapt the original classes from Treebank (Marcus et al., 1993) into 38 classes by combining punctuations (“,” “.” “:” “”” “”\`” “-LRB-” “-RRB-”) into one class and symbols (“#” “\$” “SYM”) into another. Other classes can be found in the original documentation.

Training setup. From NLTK (Loper & Bird, 2002), we use sections 0-18 for training and section 19 for validation, comprising 3,796 and 118 sentences, respectively. We retain the linear probe model weights that achieve the highest validation accuracy over 75 training epochs with a learning rate of 0.0001.

Training results. Figure 6 shows the training and validation curves of the linear probe for multiple language models. The linear probe consistently achieves high validation accuracy above 90% across models.

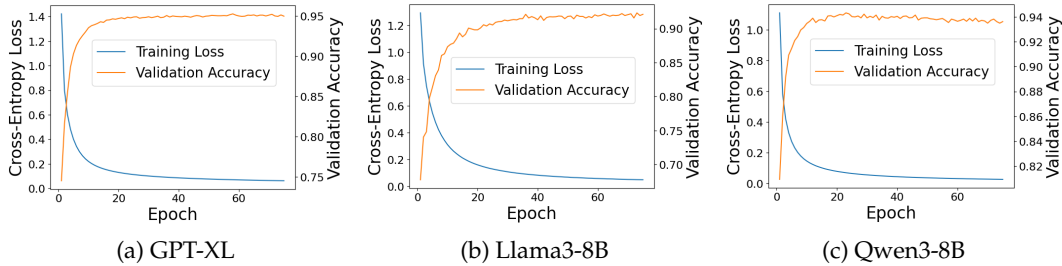


Figure 6: Training and validation curves for the linear probe under various language models. The linear probe consistently achieves high validation accuracy across models.

B Hyperparameters

For our experiments in Section 4, we evaluate SALA on top of each token truncation strategy. The hyperparameter values of the truncation strategies are selected based on previous literature and listed in Tables 6 and 7.

	GPT2-XL	Llama2-7B	Llama3-8B	Qwen2.5-7B
Top- k (k)	20	20	20	20
Top- p (p)	0.95	0.95	0.95	0.95
Typical (τ)	0.92	0.99	0.99	0.99
Min- p (p)	0.1	0.1	0.1	0.1
η -sampling	0.0009	0.0006	0.006	0.006
Adaptive (ϵ)	0.001	0.001	0.001	0.001
Top- $n\sigma$ (n)	1.0	1.0	1.0	1.0

Table 6: Hyperparameter values of the truncation strategies for the WritingPrompts dataset.

	GPT2-XL	Llama2-7B	Llama3-8B	Qwen2.5-7B
Top- k (k)	15	20	20	20
Top- p (p)	0.95	0.95	0.95	0.95
Typical (τ)	0.99	0.95	0.95	0.95
Min- p (p)	0.1	0.1	0.1	0.1
η -sampling	0.004	0.006	0.006	0.006
Adaptive (ϵ)	0.001	0.001	0.001	0.001
Top- $n\sigma$ (n)	1.0	1.0	1.0	1.0

Table 7: Hyperparameter values of the truncation strategies for the WikiText dataset.

C Prompts

C.1 Instruction-following Prompt on WritingPrompts Dataset

Section 4.3 evaluates instruction-tuned models with and without SALA for story generation. The prompt is the following:

System: You are a writer. Write a story that continues from the given prefix.
User: Prefix: <prefix>

C.2 Instruction-following Prompt for Math Reasoning

Section 4.4 evaluates mathematical question-answering datasets on Llama3.2-3B-Instruct with and without SALA. From Tang et al. (2025), the prompt is the following:

System: You are a helpful expert problem solver. Please strictly follow the user’s instructions, especially the output format.
User: Please answer the following question:\n\n<question>\nYour response *must* end with “The final answer is (answer)”. No units. For example: (Question and your reasoning) The final answer is 33.

D SALA Implementation

Below is an implementation of SALA compatible to the Hugging Face Transformers library.

```

1 class SALALogitsWarper(LogitsWarper):
2     def __init__(self, coef, embed_matrix, linear_probe, device, m=3000):
3         self.coef = coef
4         self.embed_matrix = embed_matrix
5         self.linear_probe = linear_probe
6         self.m = m
7
8     def __call__(self, input_ids, scores, outputs):
9         # Last hidden state at final time step
10        last_hidden_state = outputs.hidden_states[-1][:, -1, :]
11
12        # Top-m candidate selection
13        top_m_values, top_m_indices = torch.topk(scores, self.m, dim=-1)
14        token_probs = F.softmax(top_m_values[0], dim=0)
15
16        with torch.no_grad():
17            prefixes = last_hidden_state.repeat(top_m_indices.size(1), 1)
18            token_embeds = self.embed_matrix(top_m_indices[0])
19            full_embeddings = torch.cat([prefixes, token_embeds], dim=1)
20            pos_logits = self.linear_probe(full_embeddings)
21
22        pos_tags = candidate_pos_logits.argmax(dim=-1)
23
24        # POS-conditioned entropy adjustment
25        nentropy = self._normalized_entropy(top_m_values.view(-1))
26        adjustments = self._compute_weights(pos_tags).to(device)
27        scores[0, top_m_indices[0]] += (self.coef * nentropy *
28            adjustments)
29        return scores
30
31    def _compute_weights(classes, TAGNUMS=38):
32        # unweighted counts
33        counts = torch.zeros(TAGNUMS, device=classes.device)

```

```

33     counts = counts.scatter_add(0, classes, torch.ones_like(classes,
34                               dtype=torch.float))
35     class_probs = counts / counts.sum()
36     token_class_probs = class_probs[classes]
37     zero_mask = (token_class_probs == 0)
38     weights = torch.sqrt(-torch.log2(token_class_probs + 1e-12))
39     weights = torch.where(zero_mask, torch.zeros_like(weights),
40                           weights)
41     return weights
42
43 def _normalized_entropy(logits, dim=-1):
44     log_p = torch.log_softmax(logits, dim=dim)
45     p = log_p.exp()
46     H = -(p * log_p).sum(dim=dim)
47     return H / torch.log(torch.tensor(logits.size(dim), device=logits
48                                     .device))

```

E More Results

The experimental results for GPT-XL and Llama3-8B were shown in Table 1. Table 8 and 9 show the results for Llama2-7B, Qwen2.5-7B, and Llama3.1-70B models.

F Qualitative Analysis

We illustrate SALA with three input examples on GPT2-XL and Llama3-8B by plotting the top-10 token probability distributions before and after the logit adjustment in Figure 7. With the prefix “Donald”, the models are highly confident in the next token, resulting in low entropy and minimal adjustment. For the prefix “They”, the candidate tokens are more diverse, leading to more noticeable changes in the probabilities. With the prefix “Time”, the models exhibit low confidence in the next token, causing more pronounced probability adjustments across tokens.

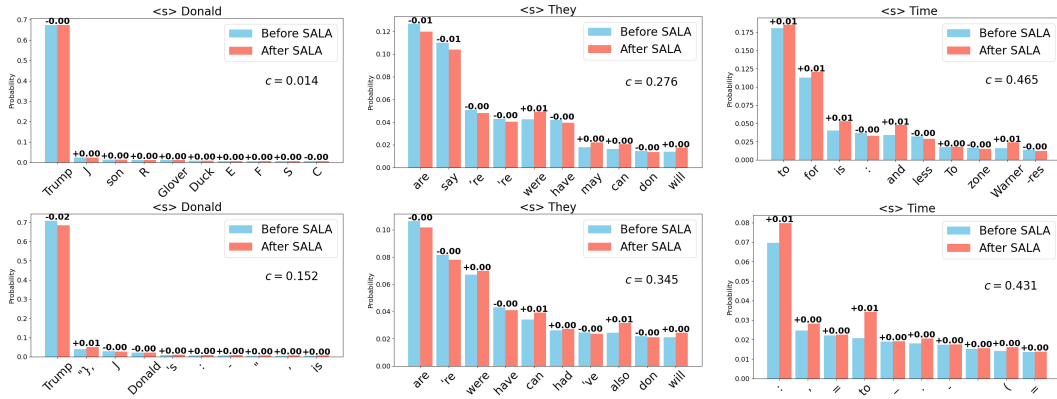


Figure 7: Examples of SALA under GPT-XL (top row) and Llama3-8B (bottom row).

	WritingPrompts					WikiText				
	R2	R3	R4	Div.	Mauve	R2	R3	R4	Div.	Mauve
Human	3.99	1.08	0.54	94.74	–	4.37	1.17	0.57	94.74	–
Top- k	12.63	7.35	5.37	80.42	21.63	11.27	5.75	3.69	81.99	5.87
+ SALA	11.39*	6.27*	4.45*	82.42*	22.71*	11.13	5.70	3.66	82.24*	6.05*
Top- p	6.29	3.24	2.32	90.36	23.86	8.95	4.37	2.73	85.70	5.49
+ SALA	5.72*	2.96*	2.14	91.18*	24.53	8.80	4.40	2.85	85.87	6.09
Typical	4.97	2.54	1.85	92.19*	22.37	7.87	3.81	2.40	87.38*	5.93
+ SALA	5.53	2.65	1.84	91.46	25.38*	8.11	3.85	2.41	87.06	6.85*
Min- p	32.19	25.61	22.14	51.23	15.86	19.62	13.07	9.96	68.84	4.36
+ SALA	28.13*	21.60*	18.36*	57.11*	16.30*	18.73*	12.29*	9.30*	70.32*	4.86*
η	6.26	3.08	2.11	90.34	25.55	10.88	5.66	3.69	82.60	5.08
+ SALA	5.49*	2.60*	1.81	91.58*	26.44	10.49*	5.41	3.52	83.28*	5.71*
Adaptive	9.55	5.25	3.80	85.29	21.81	57.43	52.68	49.59	26.40	1.72
+ SALA	8.90*	4.77*	3.42*	86.32*	23.41	57.98	53.39	50.31	26.15	2.13*
Top- $n\sigma$	22.53	16.47	13.64	65.67	19.92	16.45	10.31	7.60	73.82	4.46
+ SALA	18.48*	12.64*	10.20*	71.67*	20.67*	15.64*	9.58*	6.98*	75.20*	4.74
Human	7.72	1.93	0.68	90.05	–	8.23	2.07	0.73	90.05	–
Top- k	8.58	3.29	1.91	87.75	61.52	11.89	7.13	5.01	79.04	3.49
+ SALA	8.20	3.02	1.73	88.45*	67.23*	11.63	6.90	4.83	79.64*	4.00*
Top- p	4.94	1.59	0.90	93.16*	64.82	11.14	6.94	5.10	79.85	3.37
+ SALA	5.56	1.85	1.07	92.27	66.08*	11.11	6.87	5.03	79.89	3.81*
Typical	5.08	1.68	0.97	92.94*	62.54	11.22	7.05	5.21	79.68	3.26
+ SALA	5.32	1.68	0.95	92.67	66.22*	11.19	6.96	5.12	79.91	3.51*
Min- p	19.46	12.14	9.24	69.83	53.98	14.65	9.83	7.56	73.32	2.93
+ SALA	17.72*	10.46*	7.76*	72.83*	63.41*	14.72	9.93	7.66	73.36	3.05*
η	6.94	2.49	1.43	90.21	62.37	12.14	7.59	5.55	78.29	3.44
+ SALA	6.80	2.40	1.37	90.47	66.63*	11.86	7.32	5.29	78.84	3.75*
Adaptive	6.86	2.39	1.34	90.39	62.46	25.52	21.06	18.74	59.06*	2.31
+ SALA	6.18*	2.02*	1.11*	91.47*	68.79*	27.69	23.53	21.28	56.52	2.65
Top- $n\sigma$	12.96	6.58	4.51	80.55	59.65	13.45	8.81	6.70	75.44	2.81
+ SALA	12.23*	6.10*	4.22	81.84*	63.42*	13.37	8.73	6.62	75.74*	3.06

Table 8: Performance of different truncation strategies with and without SALA for Llama2-7B (**top**) and Qwen2.5-7B (**bottom**) on two datasets. Each sample is sampled five times, and the overall average score is reported. Higher values are highlighted in bold, and * denotes statistical significance (p -value < 0.05) compared to the baseline. Div. stands for Diversity.

	WritingPrompts					WikiText				
	R2	R3	R4	Div.	Mauve	R2	R3	R4	Div.	Mauve
Human	4.24	1.14	0.57	94.40	–	5.88	1.45	0.53	92.39	–
Top- k	10.79	6.05	3.94	82.48	2.37	9.08	4.54	2.95	85.74	5.13
+ SALA	9.90	5.35	3.38	84.03	2.47	8.69	4.23	2.73	86.49	4.99
Top- p	8.42	4.47	2.79	86.40	2.64	7.46	3.50	2.18	88.45	5.48
+ SALA	7.95	4.65	3.16	87.03	2.73	7.06	3.70	2.54	88.91	8.14*
Typical	6.94	3.47	2.12	88.99	2.34	6.38	2.82	1.70	90.24	5.69
+ SALA	7.18	3.64	2.26	88.54	2.68	6.31	2.78	1.70	90.25	5.80
Min- p	19.20	13.45	10.19	68.15	2.18	14.77	9.40	7.01	75.89	3.84
+ SALA	18.03*	12.49*	9.31*	69.80*	2.14	14.57	9.22	6.87	76.38	3.95
η -sampling	10.01	5.51	3.48	83.70	2.40	8.85	4.41	2.86	86.13	5.15
+ SALA	9.67	5.27	3.38	84.40	2.54	8.29	3.95	2.48	87.11	5.16
Adaptive	40.61	35.33	31.15	39.41	1.13	27.61	22.03	18.89	56.33	2.32
+ SALA	40.61	35.33	31.15	39.41	1.13	27.61	22.03	18.89	56.33	2.32
Top- $n\sigma$	19.43	13.67	10.34	67.65	2.31	14.47	9.13	6.75	76.39	3.94
+ SALA	18.03*	12.40*	9.24*	69.77*	2.39	13.70	8.33	6.01	77.66	3.93

Table 9: Performance of different truncation strategies with and without SALA for Llama3.1-70B on two datasets. Each sample is sampled five times, and the overall average score is reported. Higher values are highlighted in bold, and Div. stands for Diversity.