

## 853 A Random Non-Canonical Tokenization

854 In this section, we provide our algorithm for producing a random non-canonical tokenization, and a  
855 proof that each non-canonical tokenization that is *more fine-grained* than the canonical one has equal  
856 probability of being output.

### 857 A.1 Algorithm

858 We will split each token in a canonical tokenization into smaller tokens (that each exists in the  
859 tokenizer’s vocabulary). We formulate our problem as: Given a valid token  $t$ , and a set of vocabulary  
860  $\mathcal{V}$ , construct a sequence of tokens  $seq$  using tokens that exist in  $\mathcal{V}$  and form  $t$  when concatenated  
861 together. We produce  $seq$  using a recursive algorithm. Since there can be many possible  $seq$  for  
862 each  $t$ , we need to randomly choose one and guarantee that each possible  $seq$  is chosen with equal  
863 probability. We achieve this by considering recursion as producing a tree. Each path down the tree  
864 corresponds to one possible way to segment  $t$ . Each node of the tree represents a segmentation state  
865 where we have chosen some number of sub-tokens. At each node, we weigh the choice of which  
866 child node to visit by the number of leaves in the sub-tree that is rooted at each child node. This  
867 guarantees that each path down the tree is chosen with equal probability since the number of paths  
868 down a tree is equal to the number of leaf nodes in that tree. The pseudocode for the algorithm is in 1.

---

**Algorithm 1** Random Token Segmentation

---

```
1: function COUNTSEGMENTS(start)                                ▷ Cached (using memoization)
2:   if start =  $|token|$  then
3:     return 1                                                    ▷ Reached end; valid segmentation
4:   end if
5:   total  $\leftarrow$  0
6:   for end  $\leftarrow$  start + 1 to  $|token|$  do
7:     substring  $\leftarrow$  token[start : end]
8:     if substring  $\in$  vocabulary then
9:       total  $\leftarrow$  total + COUNTSEGMENTS(end)
10:    end if
11:  end for
12:  return total
13: end function
14: function BUILDSEGMENTS(start)
15:   if start =  $|token|$  then
16:     return  $\emptyset$                                               ▷ Empty segmentation
17:   end if
18:   validSegments  $\leftarrow$  []
19:   weights  $\leftarrow$  []
20:   for end  $\leftarrow$  start + 1 to  $|token|$  do
21:     substring  $\leftarrow$  token[start : end]
22:     if substring  $\in$  vocabulary then
23:       segCount  $\leftarrow$  COUNTSEGMENTS(end)
24:       if segCount > 0 then
25:         Append substring to validSegments
26:         Append segCount to weights
27:       end if
28:     end if
29:   end for
30:   if validSegments is empty then
31:     return  $\emptyset$ 
32:   end if
33:   chosenSegment  $\leftarrow$  weightedRandomChoice(validSegments, weights)
34:   return [chosenSegment] || BUILDSEGMENTS(start +  $|chosenSegment|$ ) ▷ Concatenate
    chosen segment with segmentation of remaining token
35: end function
36: procedure SEGMENTTOKEN(token, vocabulary)
37:   if COUNTSEGMENTS(0) = 0 then
38:     return  $\emptyset$                                               ▷ No valid segmentation exists
39:   else
40:     return BUILDSEGMENTS(0)
41:   end if
42: end procedure
```

---

**A.2 Proof**

**Goal:** To prove that the random segmentation algorithm chooses one valid segmentation from all possible valid segmentations with uniform probability.

**Notation:** Let  $W(i)$  denote the number of valid segmentation completions (i.e., the number of leaves in the recursive tree) for the substring starting at index  $i$ . In particular,  $W(|token|) = 1$ . Note that  $W(i)$  is calculated by the memoized recursive function *countSegments*( $i$ ), which calculates the number of leaves of the subtree rooted at  $i$ .

**Base Case:** Consider the node corresponding to  $i = |token|$  (the end of the token). Here, there is exactly one valid segmentation (the empty segmentation), so the algorithm returns it with probability

878 1. That is, every segmentation (in this case, the only one) is chosen with probability

$$\frac{1}{W(|token|)} = \frac{1}{1} = 1.$$

879 Thus, the base case holds.

880 **Inductive Hypothesis:** Assume that for any node corresponding to an index  $j$  with  $j > i$  (i.e.,  
881 deeper in the recursion tree), every complete segmentation (leaf) in the subtree rooted at  $j$  is chosen  
882 with probability

$$\frac{1}{W(j)}.$$

883 **Inductive Step:** Now consider a node corresponding to index  $i$  (with  $i < |token|$ ). Suppose  
884 that from  $i$  there are  $k$  valid branches corresponding to choosing substrings that end at indices  
885  $j_1, j_2, \dots, j_k$ , where for each  $j$  we have  $i < j \leq |token|$  and the substring  $token[i : j]$  is in the  
886 vocabulary. By definition,

$$W(i) = \sum_{r=1}^k W(j_r).$$

887 The algorithm selects the branch from  $i$  to a specific child  $j$  with probability

$$P(i \rightarrow j) = \frac{W(j)}{W(i)}.$$

888 Once branch  $i \rightarrow j$  is chosen, by the inductive hypothesis every complete segmentation (leaf) in the  
889 subtree rooted at  $j$  is chosen with probability

$$\frac{1}{W(j)}.$$

890 Thus, the probability  $P(S)$  of obtaining a particular complete segmentation  $\mathcal{V}$  that starts at  $i$  by first  
891 taking the branch  $i \rightarrow j$  and then following a specific path in the subtree rooted at  $j$  is

$$P(S) = \frac{W(j)}{W(i)} \cdot \frac{1}{W(j)} = \frac{1}{W(i)}.$$

892 Since the factor  $W(j)$  cancels, the probability  $P(S)$  is independent of the particular child  $j$  chosen.

893 **Conclusion:** By the inductive step, every complete segmentation (leaf) in the subtree rooted at any  
894 index  $i$  is chosen with probability  $\frac{1}{W(i)}$ . In particular, when  $i = 0$  (the start of the token), every valid  
895 segmentation of the entire token is selected with uniform probability  $\frac{1}{W(0)}$ . This completes the proof.

## 896 B Evaluation Details

### 897 B.1 General benchmarks

898 For short-answer benchmarks, the system prompt is:

899 You are a helpful assistant.

900 For multiple-choice benchmarks, the system prompt is:

901 You are a helpful assistant. For the following multiple choice questions,  
902 return the answer only, without any additional reasoning or explanation.

903 **MATH** MATH is a dataset composed of fairly difficult, competition level math problems [22]. The  
904 test set is composed of short answer problem that describe some scenario and asks the model to  
905 output a mathematically correct answer.

906 **GSM8K** GSM8K is a dataset consisting of relatively simple math questions that would appear  
907 in grade school math exams [14]. For GSM8K, the evaluations were done in the same manner as  
908 MATH.

909 **MMLU** MMLU is a benchmarks comprising of multiple choice questions from a wide variety of  
910 subjects. [21] We sampled 500 questions from MMLU for our evaluation. We instructed the model to  
911 only output one answer to each question without any explanation.

912 **Alpaca Eval** Alpaca Eval is an evaluation benchmark where generations from language models  
913 against given prompts are compared and judged by an annotator model. [16] The metric used was  
914 raw winrate of the perturbed model as judged by a language model. The annotator we used was  
915 *alpaca\_eval\_gpt4*, which has been shown to have the highest Spearman and Pearson correlation  
916 coefficient with human annotators.

917 **ARC Challenge and ARC Easy** Contains multiple choice questions with four options each, taken  
918 from grade school science exams [13]. ARC Easy is tests basic science knowledge while ARC  
919 Challenge requires some procedural reasoning.

920 **BoolQ** Contains true or false questions along with a context passage that provides the answer to the  
921 question. [11]

922 **CommonsenseQA** Contains multiple choice questions with five options each that requires common  
923 sense knowledge to answer. [53]

924 **COPA** Contains multiple choice questions with two options each that tests knowledge of cause and  
925 effect. [46]

926 **CUTE** Contains questions that require the model to manipulate sentence-level, word-level, and  
927 character-level structure for strings. [17]

928 **DROP** contains questions that potentially require reasoning multiple pieces of information present  
929 in a given passage. [15]

930 **HellaSwag** contains multiples choice questions with four options each that asks for the most natural  
931 continuation to some given context. [62]

932 **JeopardyQA** contains short answer questions from the “Jeopardy!” game show. [28]

933 **OpenbookQA** contains multiple choice questions with four options each that require some multi-  
934 step and common sense reasoning. [37]

935 **PIQA** contains multiple choice questions that require reasoning about the physical world. [6]

936 **TriviaQA** contains short answer questions that requires knowledge of the world. [27]

937 **Winograd** contains multiple choice questions with two options that asks to determine what a pro-  
938 noun might refer to. Answering these questions require knowledge of common sense and surrounding  
939 context. [32]

940 **Winogrande** contains questions in the same format of Winograd but there are more questions and  
941 the questions are harder. [47]

942 **TOFU** contains general short answer questions that tests the model’s ability to process world  
943 knowledge. This is the retain set of the task of fictitious unlearning dataset. [35]

944 **WikidataQA** require models to complete factual statements. [5]

## 945 **B.2 Constructed Benchmarks**

946 In this section, we provide more detail on how datasets we use in §3 are constructed.

Table 5: System prompt for tasks in §3. See Table 5 for example instructions.

---

**Counting characters:** You are a helpful assistant. The following prompt will ask you to return a sequence of words. Only return the sequence, separated by spaces. Do not provide any additional text or explanation.

---

**Common Morphemes:** You are a linguistic assistant trained to analyze lists of words and identify common morphemes. When given a list of English words, you will:

- 1) Identify the common morpheme shared by the words.
  - 2) Choose the correct option (A, B, C, or D) from the provided choices.
- Your response must be a single letter: A, B, C, or D. Do not provide any additional text, explanation, or formatting unless explicitly requested.
- 

**Code Description:** You are a programming assistant trained to analyze and interpret code snippets. When provided with a code snippet and a set of answer choices (A, B, C, or D), your task is to evaluate the code, determine its behavior, and select the answer that best describes this behavior. Your response must be a single letter: A, B, C, or D. Do not provide explanations or additional text unless explicitly requested.

---

**Arithmetic:** You are a computational assistant trained to evaluate arithmetic operations. When provided with an arithmetic expression, calculate the result and round it to the nearest integer. Respond only with the rounded result, without any additional text or explanation.

---

947 **Count Characters Task** The prompt asks the model to count the number of occurrences of a  
 948 given character in a 10-character word; we always use the most frequently occurring character.  
 949 Evaluation was done, similar to GSM and MATH, by finding the last number in the generated  
 950 response. Generations without any numbers are considered incorrect.

951 **Generate Acronym Task** The model is asked to generate a sequence of words whose first letters  
 952 form a randomly sampled five character string. For evaluation, we take the first character of each  
 953 whitespace-delimited word and check if it matches the desired acronym.

954 **Common Morpheme Task** We source data from colingoldberg/morphemes, which contains  
 955 English morphemes with example words that contained them. We generate multiple-choice questions  
 956 by pairing a particular morpheme with four random words that contain it, and then sampling three  
 957 other morphemes from the same dataset that are not contained by those four words.

958 Evaluation was performed with a regex search of the valid answer choices in the generated response.  
 959 In the extremely rare case that there are multiple instances of the letter A, B, C, or D in the generate  
 960 response, the last generated character was taken as the model’s response. The metric used was exact  
 961 match against ground truth.

962 **Codeline Description Task** The model is asked to comprehend a piece of code and choose the best  
 963 description from four options.

964 **Arithmetic Task** The model is asked to perform addition or subtraction with 10 digit numbers.  
 965 We use regex to extract numbers from the generation, which are then compared to the ground truth  
 966 answer.

### 967 **B.3 Metrics of generation quality**

968 Here we provide additional details on the metrics defined in §4.1.

969 **Spelling** We use the top 10000 most frequently appearing English words in Google’s trillion word  
 970 corpus. We only consider words with more than one character. This is because sometimes base  
 971 models will repeatedly generate the same letter, and since all English letters are in the word list, the  
 972 generation would receive a high score.

Table 6: Data format of ablations in §4.2.

<b>No ablation:</b>	<code>&lt; user &gt;Provide a detailed analysis of Candace Parker’s defensive techniques in her recent games, excluding the words "aggressive" and "blocking", in the format of a sports commentary script. &lt; assistant &gt;[Sports Commentary Script]</code> [Opening Scene...]
<b>QA Template:</b>	Question: Provide a detailed analysis of Candace Parker’s defensive techniques in her recent games, excluding the words "aggressive" and "blocking", in the format of a sports commentary script. Answer: [Sports Commentary Script] [Opening Scene...]
<b>Removing the chat template:</b>	Provide a detailed analysis of Candace Parker’s defensive techniques in her recent games, excluding the words "aggressive" and "blocking", in the format of a sports commentary script. [Sports Commentary Script] [Opening Scene...]
<b>Removing the instruction:</b>	<code>&lt; user &gt;[Sports Commentary Script]</code> [Opening Scene: A packed basketball arena, with fans eagerly awaiting the analysis of Candace Parker’s recent performances on the court.] Commentator 1: Welcome back, basketball fans! < assistant >Tonight, we’re diving into the defensive prowess of Candace Parker...

**Grammaticality** One drawback with this evaluation method is that oftentimes the model would repeat the same letter over and over again, or start counting numbers. In both of these cases, there are no detected grammar mistakes, however they are still obviously gibberish. Therefore, we only calculate grammaticality scores for generations that receive a score  $\geq 0.5$  on spelling; otherwise, we give it a grammaticality score of 0.

**Win rate** Similar to evaluation in §2, we also used `alpaca_eval_gpt4` as the evaluator and report raw win rate. In 4.1, the win rate is calculated against generations conditioned on input with canonical tokenization. In 4.2, the win rate is against generations from the **No Ablation** setting when also given character-level tokenization. By construction, the win rate of the **No Ablation** setting itself is 50%.

## B.4 Ablation Settings

For ablations on the data format, see examples of formatted data in Table 6. Our finetuning code was forked from `allenai/open-instruct`.

## B.5 Disentangling understanding from generation

For these tasks, we use 500 words randomly sampled from Google’s 10000 English word list<sup>5</sup>.

**Word Repeat** An example prompt is shown below.

Repeat each word directly, while correcting any typos.

Question: guarantees

Answer: guarantees

Question: revelation {Character-level tokenization}

Answer:

**Identifying Misspellings** We obtain the misspelled word by randomly adding, removing, or substituting a single character from the word. An example prompt is shown below.

Question: Which of the two words contains a misspelling? Respond directly

<sup>5</sup><https://github.com/first20hours/google-10000-english/blob/master/google-10000-english.txt>

998 with the answer option.  
999  
1000 Question:  
1001  
1002 A. guarantees  
1003 B. garantees  
1004  
1005 Answer: B  
1006  
1007 {9 more in context examples}  
1008  
1009 Question:  
1010  
1011 A. farmer {Character-Level tokenization}  
1012 B. farme {Canonical tokenization}