Homework 3: Tokenization, Long Context, and Pretraining Data — Solutions

CS 388: Natural Language Processing • Fall 2026

Corpus: `talking` (8), `talked` (5), `walk` (6), `walking` (3), `walks` (4), `king` (2), `thing` (3). Words are split into characters plus an end-of-word symbol `_`; ties are broken alphabetically (first symbol, then second), with `_` after every letter. Below, a segmentation is written as a list of tokens separated by spaces, e.g. `[walk ing_]`.

1. Question 1

(a) Initial vocabulary, 9 merges, final vocabulary size, training-word segmentations.

Solution. The initial vocabulary is the 12 letters occurring in $\mathcal{D}$ plus `_`:

$$\{\texttt{a}, \texttt{d}, \texttt{e}, \texttt{g}, \texttt{h}, \texttt{i}, \texttt{k}, \texttt{l}, \texttt{n}, \texttt{s}, \texttt{t}, \texttt{w}, \texttt{_}\}, \qquad \text{size } 13.$$

The merges are:

Step	Merged pair	Count	Tied with
1	`(a, l)`	26	`(l, k)`
2	`(al, k)`	26	—
3	`(g, _)`	16	`(i, n)`, `(n, g)`
4	`(i, n)`	16	`(n, g_)`
5	`(in, g_)`	16	—
6	`(t, alk)`	13	`(w, alk)`
7	`(w, alk)`	13	—
8	`(talk, ing_)`	8	—
9	`(walk, _)`	6	—

Counts for the early steps: `(a, l)` occurs in every word containing `talk` or `walk`, 8+5+6+3+4 = 26; `(g, _)` occurs in every word ending in `ing`, $8+3+2+3=16$; `(t, alk)` occurs in `talking` and `talked`, $8+5=13$, and `(w, alk)` in the three `walk` words, $6+3+4=13$. After merge 8, `(walk, _)` (6) beats `(d, _)`, `(e, d)`, `(talk, e)` (all 5).

The final vocabulary has $13+9=22$ symbols. The training words are segmented as

`talking`	`[talking_]`	`walks`	`[walk s _]`
`talked`	`[talk e d _]`	`king`	`[k ing_]`
`walk`	`[walk_]`	`thing`	`[t h ing_]`
`walking`	`[walk ing_]`		

Grading note: a student who breaks ties differently but applies their rule consistently should lose only the points for the steps that differ; the final vocabulary size (22) does not depend on tie-breaking.

(b) Tokenize `walked`, `talks`, `kings`, `stalking`, `thinking`, `sled`.

Solution. Applying merges 1–9 in order:

walked	[walk e d _]
talks	[talk s _]
kings	[k in g s _]
stalking	[s talking_]
thinking	[t h in k ing_]
sled	[s l e d _]

Two worth tracing. In `kings`, merge 3 needs `g` to be followed by `_`, but here it is followed by `s`, so only merge 4 (`in`) fires and `ing_` never forms. In `stalking`, merges 1–2 build `alk`, merge 6 attaches the `t` to give `talk`, and merge 8 then glues on `ing_`; nothing ever attaches the leading `s`.

(c) Which are tokenized well, and which poorly?

Solution. *Well:* `walked` and `talks` split into a known stem plus the suffix characters, so the token boundary lines up with the morpheme boundary. (The suffixes `-ed` and `-s` are not single tokens, but that is only a matter of efficiency.)

Poorly:

- `kings`: the suffix `ing` was only ever seen word-finally, so BPE learned `ing_` rather than `ing`; the end-of-word marker makes the word-internal `ing` unrecognizable.
- `stalking`: the segmentation `s` + `talking` cuts across the real morphology (`stalk` + `ing`), because BPE is purely frequency-driven and `talk`/`talking` were frequent in training while `stalk` never occurred; merges are applied greedily in learned order, with no notion of morphemes.
- `thinking`: `think` never appears in the corpus, so the stem shatters into characters (`t h in k`); only the suffix is recognized. Also acceptable: noting that `in` inside `think` is merged for an unrelated reason.
- `sled`: none of its character pairs were frequent in $\mathcal{D}$, so it falls back to characters. This is not *wrong*, but it is the least efficient possible encoding.

Full credit for correctly classifying the words and giving a sensible cause for at least the clearly bad ones (`kings`, `stalking`, `thinking`).

2. Question 2

(a) Length generalization from 1024 to 2048.

Solution. (i) Sinusoidal. The encoding is a fixed function, so unlike a learned embedding table the model can at least *compute* p_i for $i > 1024$; nothing crashes. But the model has only ever seen the 1024 vectors $p_1, \ldots, p_{1024}$. The low-frequency coordinates $\sin(i\omega_k), \cos(i\omega_k)$ (small ω_k) move slowly with i, and for $i > 1024$ they take on values, and combinations of values across coordinates, that never appeared in training. Positions 1025–2048 are therefore out-of-distribution inputs to every layer, and since the model learned position-specific behaviour on absolute positions, we expect predictions there to degrade badly (perplexity blows up shortly past the training length).

(ii) RoPE. Scores depend only on the offset $m-n$, so any query–key pair within 1023 positions of each other looks exactly like something seen in training; this is a real improvement, and local behaviour transfers. However, a query at position 2048 now also attends to keys at offsets 1024 to 2047, which were never seen. For the low-frequency rotation dimensions, the rotation angle $\theta_k(m-n)$ goes beyond the range covered in training, so those score contributions are out of distribution. In practice RoPE models also degrade sharply beyond their training length unless something is done,

e.g. position interpolation (rescale positions by 1024/2048 so all angles stay in the trained range), NTK-aware scaling or YaRN, or a short fine-tune at the longer length.

Key point for grading: (i) is out of distribution in *absolute position*; (ii) fixes absolute position but is still out of distribution in *relative offset* / rotation angle. Neither generalizes cleanly.

(b) Attention dilution.

Solution. **(i)** Using the lower bound with $e^5 \approx 148.41$:

n	lower bound on $\alpha_{i^\star}$
128	$148.41/275.41 \approx 0.539$
1024	$148.41/1171.41 \approx 0.127$
8192	$148.41/8339.41 \approx 0.018$

As $n \to \infty$ the guarantee goes to 0 (like e^{Δ}/n): a fixed margin Δ promises less and less attention on the needle. If $\alpha_{i^\star}$ itself shrank this way, the output $o = \alpha_{i^\star} v_{i^\star} + \sum_{j \neq i^\star} \alpha_j v_j$ would be dominated by an average of the $n-1$ irrelevant values: the needle's contribution is "washed out" even though the head still ranks it first.

(ii) Softmax normalizes over *all* positions. After dividing by e^s, the needle's numerator is fixed at 1, while the denominator $1 + \sum_{j \neq i^\star} e^{s_j - s}$ has one positive term per distractor. The ratio between the needle's weight and any single distractor's weight, $e^{s - s_j}$, does not depend on n, but the number of distractor terms grows with n, so the needle's *share* of the total shrinks.

(1) Distractors cannot all be pushed arbitrarily low. If every score lies in $[-B, B]$, then $s_j - s \geq -2B$, so every distractor term satisfies $e^{s_j - s} \geq e^{-2B}$, no matter how the distractor scores are distributed. The denominator is therefore at least $1 + (n-1)e^{-2B}$, which gives the matching *upper* bound

$$\alpha_{i^\star} \;\leq\; \frac{1}{1 + (n-1)e^{-2B}} \;\xrightarrow{n \to \infty}\; 0.$$

Together with the given lower bound, $\alpha_{i^\star}$ is sandwiched between two quantities that both decay like $1/n$, so it really does go to 0; the spread of the distractor scores only affects the constant.

(2) Δ cannot be made arbitrarily large. The same bound caps the margin at $\Delta \leq 2B$. q and k are linear maps of layer-normalized hidden states, so $|q^\top k|$ is bounded by quantities (the weight norms and the LayerNorm scale) that are fixed once training ends; they cannot grow with the test-time context length. Moreover, the weights were tuned for $n \leq 1024$, so the margin they produce is only as large as was needed at that length. (Making scores very large would also make the softmax nearly one-hot everywhere, hurting optimization and heads that genuinely need to average.)

Full credit for (ii) requires both points; the key step for (1) is the upper bound $\alpha_{i^\star} \leq 1/(1 + (n-1)e^{-2B})$.

(c) A fix.

Solution. Many answers are acceptable; full credit for a precise description plus an explanation tied to (b)(i). The most direct one is *length-dependent logit scaling* (e.g. "log-n scaling" / scalable softmax):

$$\alpha_i = \frac{\exp(c \log n \cdot s_i)}{\sum_j \exp(c \log n \cdot s_j)},$$

for a constant (or learned) c. The margin of at least Δ becomes a margin of at least $c\Delta \log n$, so the lower bound from (b) becomes

$$\alpha_{i^\star} \;\geq\; \frac{n^{c\Delta}}{n^{c\Delta} + n - 1},$$

which no longer goes to 0: if $c\Delta \geq 1$ the bound stays at least about $1/2$ for every n, and if $c\Delta > 1$ it goes to 1. The temperature grows with the context exactly fast enough to cancel the linear growth of the denominator.

Other acceptable answers: *sparse / top-k attention* (only the k highest-scoring keys enter the softmax, so the denominator has k rather than n terms and $\alpha_{i^\star} \geq e^{\Delta}/(e^{\Delta} + k - 1)$ for all n); *sliding-window or chunked attention combined with retrieval*, so each softmax ranges over a bounded number of positions; or alternatives to softmax such as sigmoid attention, where weights are not forced to sum to 1 and therefore do not compete with each other. An answer that just says "use a bigger model" or "train on longer sequences" without explaining the effect on the denominator gets partial credit at most.

3. Question 3

Solution. Answers will vary; there is no single correct table. Award points per row for (1) a genuine snippet of at most 400 characters, (2) the `url` field of that datapoint, and (3) a convincing reason; the three reasons must be different. Common good categories, several of which correspond to the cleaning filters used to build C4 from `en.noclean`:

- **Boilerplate / navigation text**: menus, cookie notices, "terms of use", "JavaScript must be enabled", lines without terminal punctuation. Low information, and repeated across many pages.
- **Non-natural-language content**: source code, minified JavaScript or CSS, lorem ipsum, long lists of keywords or product SKUs (SEO spam).
- **Duplicated content**: the same paragraph repeated within a page or across pages; encourages memorization.
- **Offensive or explicit content**: pornography, slurs, hate speech.
- **Personal information**: names together with phone numbers, emails, or home addresses.
- **Non-English or mixed-language text** that slipped through language ID.
- **Machine-generated or very low-quality text**: spun articles, auto-translated pages, spam comments.
- **Very short pages** that carry essentially no content.

Strong answers note *why* the content hurts the model (wasted compute, memorization and privacy risk, toxic generations, distribution mismatch), not just that it looks bad. A thoughtful answer

may also point out the cost of over-filtering: e.g. C4's bad-words filter is known to disproportionately remove text written in dialects such as African-American English and text about LGBTQ+ identities.