

Quiz 4

601.471/671 NLP: Self-Supervised Models

Johns Hopkins University Fall 2026

Name: _____

Instructor version

Time: 20 minutes. Total: 100 points + 5 possible bonus points.

Write the requested answers in the boxes. Explain Q1(c), Q2(d), and each bug in Q3; for all other parts, the requested pair, numbers, vectors, or short labels suffice. Each required part of Q1 and Q2(a)–(c) is worth 10 points; Q2(d) and Q3 are worth 20 points each. The WordPiece part of Q1(a) is optional: leaving it blank does not reduce your score; earned bonus points are added beyond 100. The 20-minute limit includes bonus work. No calculator is needed. Sequence positions are numbered from 0; log denotes the natural logarithm. Unless stated otherwise, dropout is disabled.

Q1. Tokenization and Neural Language Models

30 points

- (a) At one tokenizer-training merge step, consider only the two candidate adjacent token pairs below. Counts are measured before this merge:

Pair	Pair frequency	First token frequency	Second token frequency
(x, y)	12	40	40
(z, w)	8	8	8

Required BPE (10 points). Which of the two pairs does frequency-based BPE merge next? Write the ordered pair of tokens.

Solution: Answer: (x, y). Frequency-based BPE selects the most frequent adjacent pair among the candidates. Here $f(x, y) = 12 > f(z, w) = 8$, so it merges x and y into a new token xy . Individual token frequencies do not enter this BPE comparison; normalizing by them would change the rule.

Grading: 10 points for the correct pair; no explanation is required.

Optional WordPiece (5 bonus points; extra reading). Using the same table, apply the pair-scoring rule from the extra reading and select the higher-scoring candidate. Write the ordered pair of tokens.

Solution: Answer: (z, w). The scoring rule in the extra reading is $\text{score}(u, v) = f(u, v) / (f(u)f(v))$. Thus $\text{score}(x, y) = 12 / (40 \cdot 40) = 3/400$, whereas $\text{score}(z, w) = 8 / (8 \cdot 8) = 1/8 = 50/400$. The latter is larger, so WordPiece selects zw even though its raw pair count is lower. The denominator discounts individually common components. This is the reading's simplified rule.

Grading: 5 bonus points for the correct pair; no explanation is required. A blank or incorrect answer costs no required points.

- (b) The input embedding table and output layer use separate parameters. The table is used only for lookup. One backward pass looks up IDs [2, 5, 2]; the loss gradients with respect to the three returned vectors are [2, 1], [0, 3], and [-1, 4], in that order. These are the only uses of the table, and there is no weight decay or special padding behavior. Write the two-component data-loss gradient for table row 2.

Solution: Answer: [1, 5]. Both occurrences of ID 2 retrieve the same trainable row $E[2]$. Backpropagation adds the gradient contributions from every use of that shared parameter: $\nabla_{E[2]} \mathcal{L} = [2, 1] + [-1, 4] = [1, 5]$. We sum rather than average: the incoming gradients already reflect the loss's reduction convention. The middle vector [0, 3] belongs to row 5 and does not contribute to row 2. IDs are addresses; we differentiate the selected floating-point entries, not the IDs.

Grading: 5 points per correct component. Handout 10 §2.2 and Check 2.

- (c) In Handout 10's Alice/Bob example, two prefixes differ in an earlier name but have the same final four tokens, the key belonged to. A deterministic fixed-window LM sees only those four tokens. With the same model parameters used for both prefixes, can adding hidden layers alone make their next-token distributions differ? Answer yes/no, explain why in one sentence, and name one input change that would expose the distinction.

Solution: Answer: No; expose the earlier name in the input. The model receives the identical four-token suffix for both prefixes. Its computation is a deterministic function of that suffix, so the same embeddings, hidden activations, logits, and next-token distribution result in both cases. Extra layers or width may change this function but cannot supply the discarded name. Expanding the window far enough to include Alice/Bob, or supplying a state carrying earlier context, makes the inputs distinguishable. This permits learning the distinction; it does not guarantee it.

Grading: 2 points for no, 4 for the identical-input reason, and 4 for a valid input change. Handout 10 §6 and Check 6.

Q2. Attention, Masking, and Cost

50 points

- (a) For one attention head at query position 1, scores for keys 0, 1, 2 are $[\log 3, 0, \log 6]$, already divided by $\sqrt{d_k}$. The corresponding value vectors are $[4, 0]$, $[0, 4]$, and $[8, 8]$; there is no padding. Before softmax, mask keys at positions later than the query; earlier positions and the query's own position are permitted. Write (i) the three attention weights in key order 0, 1, 2, including any zero weight, and (ii) their weighted sum of values, before output projection or residual addition.

Solution: Answer: Weights $[3/4, 1/4, 0]$; output $[3, 1]$. At query position $i = 1$, the mask allows keys $j \leq 1$, so key 2 must be excluded. Replace its score with $-\infty$ before softmax, giving $[\log 3, 0, -\infty]$. Exponentiating yields $[3, 1, 0]$; dividing by their sum 4 yields the weights. The retrieved vector is $(3/4)[4, 0] + (1/4)[0, 4] + 0[8, 8] = [3, 1]$. The original scores are already scaled, so do not divide by a head-width factor again. Masking after softmax without renormalizing would retain an incorrect denominator of $3 + 1 + 6 = 10$.

Grading: 5 points for the weight vector and 5 for the output vector. Handout 12 §1.

- (b) An encoder-decoder has source length $S = 5$, target length $U = 3$, batch size $B = 2$, and $H = 4$ heads. There is no padding. For standard decoder cross-attention, write four answers: the sequence of representations supplying Q; the sequence supplying K/V; the numerical weight tensor shape with axes ordered as batch, head, query position, key position; and the softmax axis, identified by its role and length.

Solution: Answer: Q from decoder rows; K/V from final encoder rows; weights $[2, 4, 3, 5]$; softmax over the last, source/key axis. Each of the 3 decoder queries retrieves information from the 5 source positions. Per head, $QK^\top$ therefore has shape 3×5 ; adding batch and head axes gives $[B, H, U, S] = [2, 4, 3, 5]$. For each batch item, head, and target query, the 5 source weights sum to 1. The entire source is available conditioning information, so cross-attention needs no triangular restriction on source positions. Decoder self-attention separately prevents each target position from reading later target positions.

Grading: 2 points for Q source, 2 for K/V source, 3 for shape, and 3 for the normalization axis. Handout 12 §3.1.

- (c) In a dense full-sequence forward pass, replace length T by $2T$, holding batch size, model width, head count, FFN width, and dtype fixed. Using leading arithmetic/storage terms, write the multiplicative growth factors for (i) Q/K/V projections, (ii) both $QK^\top$ and weighted-value products, and (iii) storage of one explicitly materialized attention-weight tensor. Give the three factors in order; do not estimate wall-clock runtime.

Solution: Answer: $(2, 4, 4)$, in the requested order. Q/K/V projections apply fixed-width linear maps separately to each token: their arithmetic is proportional to $BT D^2$, so replacing T by $2T$ doubles it. Attention scores and value mixing involve query-key pairs; across all heads their arithmetic is proportional to $BT^2 D$, so it grows by $2^2 = 4$. Explicit attention weights have BHT^2 entries and likewise quadruple. These are arithmetic and storage factors, not a claim that wall-clock runtime quadruples; kernels and the relative costs of operations matter.

Grading: 4 points for the projection factor, 3 for pairwise arithmetic, and 3 for storage. Handout 12 §4 and Check 4.

- (d) **20 points.** Consider Handout 11's proposed replacement of softmax attention by hard selection of the single highest-scoring key. Hold values fixed and make a small query/key change that leaves the unique winning key unchanged. Consider only gradients through this attention operation. Does the small query/key change alter the output? What does this imply about the gradient through the selection operation to Q/K? Separately, can the selected value receive a nonzero gradient from a loss depending on the output? Use ordinary derivatives, and explain your answers in one or two sentences.

Solution: Answer: The small query/key change does not alter the output, so the gradient through selection to Q/K is zero; the selected value can still receive a loss gradient. Write the hard-selection output as $y = v_{j^*}$, where $j^* = \arg \max_j s_j$. With a unique winner that does not change under a small score perturbation, y is constant with respect to the scores; its local score derivative is zero. Q/K therefore receive no gradient through this selection. However, y still depends directly on v_{j^*} , so $\partial y / \partial v_{j^*} = I$ and an upstream loss gradient can reach that value. Winner-switch boundaries are discontinuous.

Grading: 4 points for stating that the output is unchanged, 6 for the zero Q/K gradient with its constant-output explanation, and 10 for the selected value gradient with its direct-dependence explanation. Handout 11 §2.2, "Why not just pick the best key?"

Q3. Debug a Decoder Model

20 points

The snippet below is intended to train a decoder model to predict the next input token at each position. Each attention query may use its own position and earlier positions, but may not use later positions. There are **exactly three independent bugs** in the attention computation or loss calculation.

For each bug, give its line number, a corrected expression or line, and a one-sentence reason. Give one bug per answer box; any order is acceptable. Analyze each bug assuming the other two have been corrected. Do not count a consequence of a bug, such as an invalid numerical output, as a separate bug. You do not need to rewrite the full program.

Use these assumptions:

- `math`, `torch`, and `torch.nn.functional` as `F` are imported. All tensors and model parameters are on the same device and have compatible dtypes.
- B is batch size, H is head count, $T \geq 3$ is sequence length, and d is head width. These and `vocab_size` are positive integers. `token_ids` has shape $[B, T]$, dtype `torch.long`, and IDs in $[0, \text{vocab_size} - 1]$. There is no padding or dropout.
- `q`, `k`, and `v` each have shape $[B, H, T, d]$. They are correctly computed from each position's token and position embeddings separately; no earlier operation has mixed positions. In `scores`, the final two axes are query position, then key position.
- `lm_head` correctly maps the last feature axis of width Hd to `vocab_size` raw logits. The intended loss averages next-token cross-entropy over all $B(T - 1)$ input positions with a following token in the same sequence. All code outside this snippet is correct.

```
1 scores = q @ k.transpose(-2, -1) / math.sqrt(d)
2 allowed = torch.ones(T, T, device=q.device).tril().bool()
3 scores = scores.masked_fill(allowed, float("-inf"))
4 weights = scores.softmax(dim=-2)
5 context = weights @ v
6 context = context.transpose(1, 2).reshape(B, T, H * d)
7 logits = lm_head(context)
8 loss = F.cross_entropy(
9     logits[:, :-1].reshape(-1, vocab_size),
10    token_ids[:, :-1].reshape(-1),
11 )
```

Solution: Line 3: reversed attention mask. Replace the argument `allowed` by `~allowed`:

`scores.masked_fill(~allowed, float("-inf"))`. The lower triangle marks permitted past/current keys ($j \leq i$). `masked_fill` replaces entries where its Boolean argument is true, so the faulty line removes permitted keys and retains future keys. Its last query has every key masked, which also explains the NaNs; this is a symptom of the same mask bug, not a fourth independent bug.

Line 4: wrong softmax axis. Use `weights = scores.softmax(dim=-1)`. Scores have shape $[B, H, T_{\text{query}}, T_{\text{key}}]$. Each query needs a distribution over keys, so normalize across the last axis. The faulty code instead makes different queries compete for each key's mass.

Line 10: labels are not shifted. Use `token_ids[:, 1:].reshape(-1)`. Line 9 retains logits at input positions 0 through $T - 2$; these must predict input tokens at positions 1 through $T - 1$. The original slice pairs each row with its own input token, training current-token reconstruction rather than next-token prediction. Both corrected arguments contain $B(T - 1)$ rows. Cross-entropy correctly receives raw logits; no extra softmax is needed.

Grading: Line 3: 6 points (2 for location/defect, 2 for correction, 2 for reason). Lines 4 and 10: 7 points each (2 for location/defect, 3 for correction, 2 for reason). Accept equivalent correct expressions and any bug order. Grade each independently as if the other two were corrected. Sources: Handout 11 §3.2 and §7; Handout 12 §1.