

Midterm 1 · Solutions

CS 5090 · Fri Oct 9, 2026 · 100 points + 10 bonus · 50 minutes

Part A · Multiple choice (2 points each)

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15
c	b	b	b	b	c	b	c	b	b	b	b	a	d	b

A6: cumulative 0.45, 0.70, 0.85, so the first three. A8: $20 \times 50M = 1B$.

Part B · True or false (1 point each)

B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
T	T	F	F	T	F	T	T	F	T

Part C · Work it out

C1 (10).

- (a) The score against **t4** (the 1): t4 comes after t3, and at generation time it does not exist yet.
(b) softmax over [0, 2, 0]: $e^0, e^2, e^0 \approx 1, 7, 1$; sum = 9. Weights = **1/9, 7/9, 1/9 $\approx$ 0.11, 0.78, 0.11** for t1, t2, t3 (t4 gets 0).
(c) **t2** (the highest score dominates after exponentiation); the weights add up to **1**.

C2 (10).

- (a) $20 \times 40M =$ **800M tokens**.
(b) $p(\text{correct token}) \approx 1/16,384$, so $L = -\ln(1/16,384) = \ln 16,384 = 14 \times \ln 2 \approx 9.7$.
(c) **Too big**: 100M tokens supports only about $100M / 20 = 5M$ parameters (or: 40M would need 800M tokens).

C3 (10).

- (a) $\text{pass}@1 = 2/5 =$ **40%** (P1, P4). (b) $\text{pass}@4 = 4/5 =$ **80%** (all but P2).
(c) The model is unchanged; spending more compute at inference (more samples) solves more problems, because the model can often produce a correct solution even when its first attempt is wrong.
(d) P3: **[0, 0, 1, 0]**.

Part D · Short answers

D1 (10).

Method	Signal	Training data
SFT / instruction tuning	P	(instruction, good response) demonstrations, written by humans or synthetic
RLHF with PPO	Q	human rankings of several model responses per prompt (to train the reward model), then prompts to generate from
DPO	R	preference pairs: (prompt, chosen, rejected)
RLVR with GRPO	S	prompts whose answers can be checked automatically (maths with answers, code with unit tests)

- D2 (10).** (a) A model imitates its data, and code data contains only {problem, code} pairs: almost no examples of answering an ambiguous problem with a question. More of the same data teaches more of the same behaviour.
(b) Take existing coding problems (APPS), use an LLM to turn them into ambiguous, inconsistent, or incomplete versions, and generate the clarifying question each should trigger (~30K examples). Merge with the original {problem, code} data and instruction-tune (fine-tune) the code LLM on the combined set.

D3 (10). Layer norm normalizes each token's activation vector across its features, keeping activations in a stable range so deep stacks train faster and more stably. Batch norm normalizes across the batch, which is a poor fit for language models: variable-length sequences, small batches, and single-sequence inference where batch statistics are unstable or meaningless. Layer norm uses only the token's own features, so it behaves the same in training and inference.

Bonus (10 points)

(a) RNN: $n - 1$ steps (accept "about n "): information from token 1 has to pass through every hidden state in turn. Self-attention: **one** step, because every position attends directly to every earlier position.

(b) Any two: (1) no recurrence, so all positions of a sequence are processed **in parallel** instead of one after another; (2) the work is a few large **matrix multiplications** (QK^T , the FFN), which is what GPUs are built for; (3) no long chain of sequential steps, so gradients do not have to flow back through n time steps (the vanishing-gradient problem that made RNNs slow and hard to train on long sequences); (4) the whole batch of sequences fits one big batched operation.

(c) **Word order / position.** An RNN knows the order because it reads tokens one at a time; attention is a weighted sum over a set, so it is blind to order. The Transformer adds **position encodings** to the token embeddings (fixed sinusoidal encodings in the original paper; learned position embeddings in GPT-style models).