CSCI 5942, AI Engineering, Fall 2026

Final review

Questions from Quizzes 1 to 6. Ranking was done to prioritize the hardest and oldest content.

The causality test

Your Assignment 1 model, a character-level decoder-only transformer with a 256-character context and the 65-character vocabulary that data/prepare.py built from the 1.1M-character Shakespeare corpus. Before training, it had to pass four tests: test_output_shapes, test_causality, test_init_loss_near_uniform and test_can_overfit_tiny_batch.

A teammate rewrites CausalSelfAttention. Their version passes the shape test and the overfit test, and its validation loss falls below 0.3 nats per character within 200 iterations, well under the 1.46 the reference run reaches after 5,000. (Part (a): the most likely bug is a broken causal mask.)

(b) Which of the four tests catches it, and what does that test check? Name it and describe the check.

The causality test

The bug this is intended to catch is a broken causal mask meaning each position can see characters after it.

For instance, x = “brown fox” and x′ = “brown fax”. Positions 0 to 8 are b r o w n (space) f o x, so the only difference is position 7: o becomes a.

1. What does test_causality do?

Runs the model on two inputs that differ only at a later position, then checks that the logits at every earlier position are identical.

2. With a correct mask, position t sees positions 0 to t. Are the logits for “brown” equal in x and x′?

Yes. Positions 0 to 6 never see position 7, so their logits match. Only 7 and 8 change, so the test passes.

3. With the broken mask, every position sees every character. What happens to the logits for “brown”?

They change, because “brown” can now see the a at position 7. The equality check fails, and the test catches the bug.

4. Why do the other three tests miss it?

Shapes checks only sizes, overfit checks only that a batch can be memorized, and init loss checks only the starting loss.

The loss at the start of training

Your Assignment 1 model, a character-level decoder-only transformer with a 256-character context and the 65-character vocabulary that data/prepare.py built from the 1.1M-character Shakespeare corpus.

(a) At iteration 0 the reported loss is about 4.2 nats per character. Recall that the loss is the average over positions of −ln p(true next character), in nats. Where does that number come from? What would you suspect if the number at iteration 0 was 9 instead?

The loss at the start of training

1. How is the loss in nats per character calculated?

For each character, you take −ln of the probability the model gave the true next character, then average over all characters.

2. At initialization the logits are close to zero. What probability does each character get, and what loss does that give?

Each gets $1/65 \approx 0.015$, so the loss is $-\ln(1/65) = \ln 65 \approx 4.17$ nats.

3. The run starts at a loss of 9. What probability did the true character get, and what does that tell you?

$p = e^{-9} \approx 0.00012$, is well below 1/65. The model here starts confident and wrong indicating bad initialization.

4. Which test was written to catch this, and what does it check?

test_init_loss_near_uniform. It checks that the starting loss is within 0.35 nats of ln 65.

Whether a 100M-parameter model is worth training

Every rung of the Assignment 1 ladder trained on the same 82M characters (5,000 iterations × batch 64 × 256 characters) from the 1.1M-character Shakespeare corpus.

(b) Product asks whether a 100M-parameter version is worth the GPU budget. Based on the table, what is your response?

rung	non-embedding parameters	best val loss (nats/char)
pico	27K	2.04
nano	103K	1.75
micro	0.80M	1.47
mini	4.7M	1.48
base	10.6M	1.46

Whether a 100M-parameter model is worth training

1. The loss falls from pico to micro, then goes flat. From micro to base, how many times the parameters, for how much loss?

About 13 times the parameters for 0.01 nats.

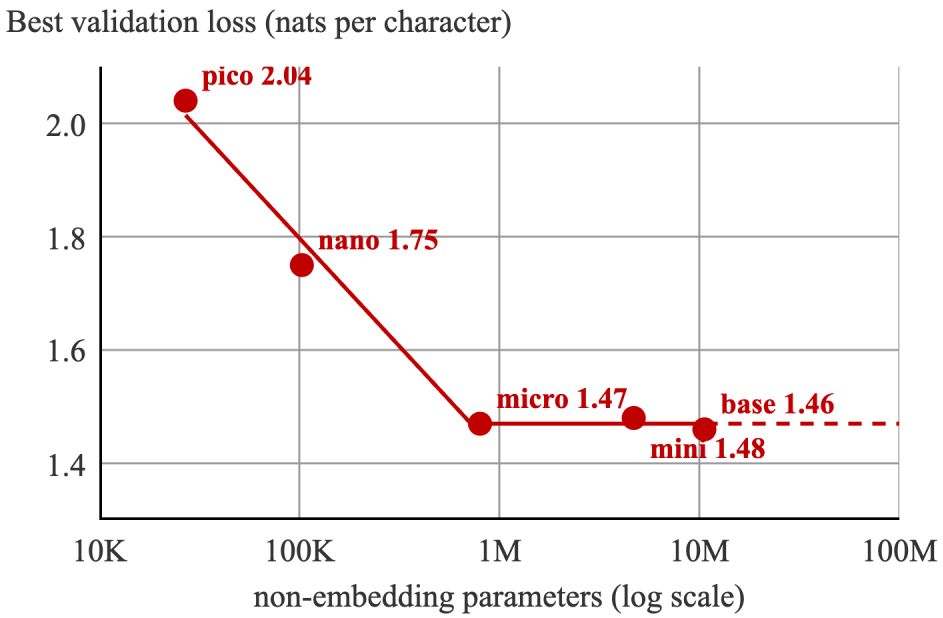
2. How many training characters per parameter does base see?

82M ÷ 10.6M ≈ 7.7 tok/param, against about 20 tok/param for compute-optimal. About 80 passes over roughly 1M unique characters.

3. What would a 100M model cost, and what would it gain?

About 9.4 times the compute for no expected gain. Spend on unique data instead.

Compute = 6 x Parameters x Number of Tokens
(C = 6ND)



Running screenplays through the Shakespeare pipeline

Your Assignment 1 model, a character-level decoder-only transformer with a 256-character context and the 65-character vocabulary that data/prepare.py built from the 1.1M-character Shakespeare corpus.

Product now wants the same system for modern screenplays and proposes fine-tuning the base checkpoint on 500 screenplays.

(a) Before any training can run, one component of the Assignment 1 pipeline gets in the way. Which component is it, and what happens when a page of a screenplay goes through it?

Running screenplays through the Shakespeare pipeline

Character classes. Each press fills in one row.

characters	in Shakespeare's 65?	in screenplays?
Uppercase letters A to Z	**yes**	**yes**
Lowercase letters a to z	**yes**	**yes**
Space and newline	**yes**	**yes**
Punctuation ! $ & ' , - . : ; ?	**yes**	**yes**
Parentheses, as in (V.O.)	**no**	**yes**
Double quotes "	**no**	**yes**
Digits 0 to 9	**only 3**	**yes**

1. Which part of the pipeline decides what characters the model can read? When was it built, and from what text?

The tokenizer. Its 65-character vocabulary was built from the Shakespeare text.

2. Fill in the table. Which classes do screenplays need that the 65 leave out?

Parentheses, double quotes and every digit except 3.

3. What happens to those characters when a screenplay is encoded? What happens if you delete them instead?

Encoding raises a KeyError at the first one.

Deleting them instead loses the parentheses, quotes and scene numbers.

Rebuilding the vocabulary

Your Assignment 1 model, a character-level decoder-only transformer with a 256-character context and the 65-character vocabulary that data/prepare.py built from the 1.1M-character Shakespeare corpus.

Product wants the system for modern screenplays and proposes fine-tuning the base checkpoint on 500 screenplays. Part (a): the 65-character vocabulary, fixed when data/prepare.py ran, has no parentheses, no double quotes and no digit other than 3.

(b) Outline the fix and what it forces you to redo, including what happens to the trained checkpoint.

Rebuilding the vocabulary

1. What changes about the vocabulary, and what must be redone to the data?

Build one vocabulary from the union of both corpora and re-encode both.

2. Which parts of the model have one row per character? How big are they before and after?

The embedding table, which the output layer shares.
It grows from 65 rows to one row for however many characters are in the new vocabulary.

3. Will the old checkpoint still load? If not, what are the options?

No, the shapes no longer match.
1) Copy the 65 old rows, initialize the new ones and fine-tune
2) Retrain from scratch.

4. What should be re-run before training starts?

The tests, with the new vocab_size. A longer block size is also worth considering (256 characters is only about 40 words).

Reading the python-code column

Teacher-forced loss in nats per character on held-out text. Characters the 65-character vocabulary cannot represent are deleted first. A uniform guess over 65 characters scores ln 65 ≈ 4.17 on any text.

(a) Product sees the 27K-parameter pico ahead of nano, micro and mini on python-code, and asks whether the smallest model should become the team's code model. What is your response?

model	non-embedding params	shakespeare	python-code
pico	27K	2.03	4.48
nano	103K	1.73	5.52
micro	0.80M	1.45	5.06
mini	4.7M	1.45	4.73
base	10.6M	1.43	4.38
characters deleted as OOV		0.0%	7.7%

Reading the python-code column

1. What does a uniform guess score?

ln 65 ≈ 4.17 nats per character (see the red dashed line on the chart).

2. What is the expected performance of any of these models?

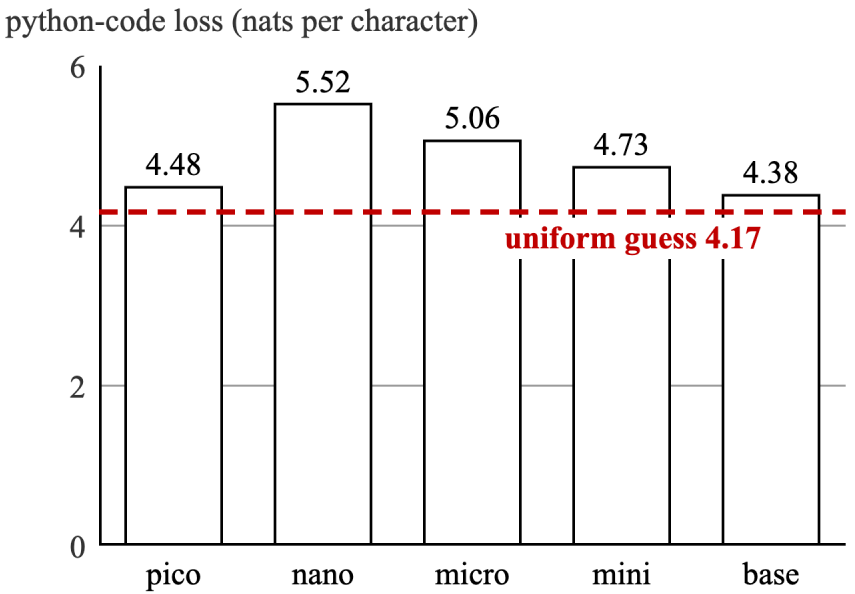
They are all worse than chance. All five sit above the 4.17 line (4.38 to 5.52), so none beats a random guess.

3. So what is this column actually ranking? Why is pico ahead?

The rank is based on how confidently wrong each model is. Pico is more than likely ahead because it is the least sure of itself.

4. The last row of the table says 7.7% of the python-code characters were deleted as out of vocabulary (0.0% for Shakespeare). What does that tell you?

Underscores, brackets, equals signs and most digits were removed before scoring, so the column scores code-shaped prose, not real code.



Whether the judge’s claims hold up when the prompt changes

Gemini scores base’s 50 samples from 1 to 10 under the Shakespeare rubric, three times: with the rubric as written, with a paraphrase that keeps the meaning and none of the wording, and with the three axes listed in a different order. Same 50 samples every row.

prompt	grammar	coherence	style_match
the assignment’s rubric	5.9 ± 0.5	3.8 ± 0.5	7.6 ± 0.4
paraphrased rubric	5.4 ± 0.5	4.1 ± 0.5	6.9 ± 0.5
axes reordered (style_match first)	6.0 ± 0.5	3.6 ± 0.5	7.5 ± 0.4

How the prompts differ: the paraphrase keeps what each axis means but changes every word; the reorder keeps the wording but asks for style_match first. The samples, the judge and the 1 to 10 scale stay the same, so any change in a score comes from how the judge reads the prompt, not from the samples.

The write-up claims: (i) base’s grammar is 5.9 out of 10; (ii) style is base’s strength and coherence its weakness.

(b) Which of the two claims survives the paraphrase and the reordering, and what makes a claim survive?

Whether the judge's claims hold up when the prompt changes

The claims: (i) base's grammar is 5.9 out of 10; (ii) style is base's strength and coherence its weakness.

1. Claim (i): does "grammar is 5.9" survive when only the wording of the prompt changes? Why or why not?

No. The same samples score 5.9, 5.4 and 6.0. Wording alone moves the score 0.6, as much as the ±0.5 interval, so 5.9 reflects the prompt as much as the model.

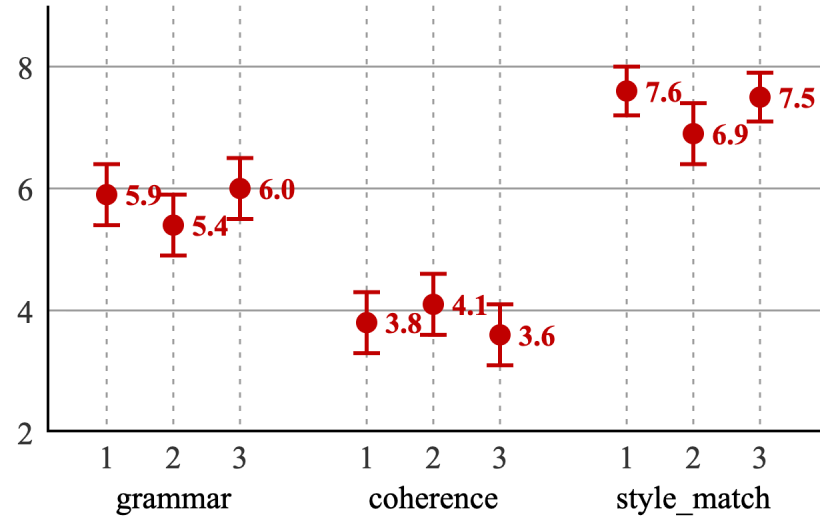
2. Claim (ii): does "style beats coherence" survive the same changes? Why or why not?

Yes. Style minus coherence is 3.8, 2.8 and 3.9. The gap is about 3 points which is much bigger than the 0.6 the wording moves things, so the order never flips.

3. What does a claim need in order to survive a change of wording?

The effect has to be larger than how much the score moves when only the wording changes.

Score out of 10. 1 = rubric as written, 2 = paraphrased, 3 = axes reordered.



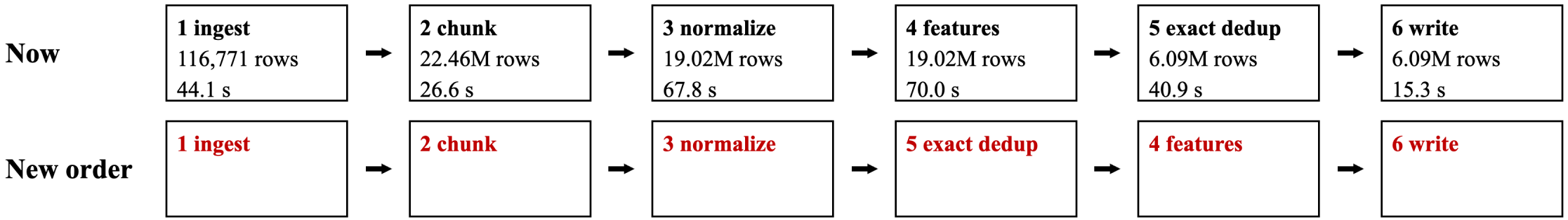
Making the data pipeline faster

Your Assignment 3 Spark pipeline turns five raw WET files (116,771 pages) into a clean, deduplicated corpus in six stages, 264.7 s in total.

(b) You will rebuild this corpus many times this week to try featurization strategies, and the runtime is too slow. Change one: can you reorder the stages and stay correct? Name the stages you would move.

stage	rows	% of lines	MB	seconds
1 ingest	116,771	n/a	1073.2	44.1
2 chunk	22,461,175	100.00%	1050.2	26.6
3 normalize	19,024,390	84.70%	684.6	67.8
4 features	19,024,390	84.70%	684.6	70.0
5 exact dedup	6,093,868	27.13%	437.6	40.9
6 write	6,093,868	27.13%	437.6	15.3

Making the pipeline faster: reorder



1. Which stage takes the most time, and how many rows does it process?

Features, which takes 70 seconds on 19.0M rows.

2. Could that stage run after a stage that removes rows, without changing the result? Fill in the new order.

Yes. The dedup step compares line text and does not use the features, so it can go first. Features then runs on 6.1M rows (about 22 s).

Making the data pipeline faster

Your Assignment 3 Spark pipeline turns five raw WET files (116,771 pages) into a clean, deduplicated corpus in six stages, 264.7 s in total.

(b) You will rebuild this corpus many times this week to try featurization strategies, and the runtime is too slow. Change two: can you avoid repeating the same work across runs? Name the stages whose output you would reuse.

stage	rows	% of lines	MB	seconds
1 ingest	116,771	n/a	1073.2	44.1
2 chunk	22,461,175	100.00%	1050.2	26.6
3 normalize	19,024,390	84.70%	684.6	67.8
4 features	19,024,390	84.70%	684.6	70.0
5 exact dedup	6,093,868	27.13%	437.6	40.9
6 write	6,093,868	27.13%	437.6	15.3

Making the pipeline faster: reuse



Same in every experiment: run once and save the deduplicated lines.

1. Which stages give the same output in every experiment?

Ingest, chunk, normalize and exact dedup. Only the features change between experiments.

2. Where would you save their output, and what does each experiment rerun?

Save (cache or materialize) the deduplicated lines once. Each experiment reads them and reruns only features and write.

Removing the hostname filter

You pretrain on mixes of your Assignment 3 web corpus at a fixed 50M-token budget and evaluate on the four fixed corpora from Assignment 2 (nats per character, lower is better).

mix	wikipedia	tinystories	shakespeare	python-code
baseline (query: true)	2.16	2.19	2.79	3.10

(a) read_wet filters certain pages by hostname from the dataset. A teammate deletes this filter and rebuilds everything. Their baseline comes back 0.2 nats better on wikipedia, while the other columns are stable. What happened and how can you tell? Is the resulting corpus a better dataset to use?

Removing the hostname filter

1. Without the filter, only the wikipedia loss improved (2.16 to 1.96). What does the hostname filter remove from the training data? How does that connect to the column that moved?

It removes Wikipedia pages. Without the hostname filter, the evaluation text gets into training.

2. Is the model actually better? What is that column measuring now?

No. The column now measures memorization (contamination). The other three columns act as the control.

Choosing dp, tp and pp

You are scaling up your Assignment 4 pretraining setup. The app needs a bigger model than anything in the assignment, and the team has four nodes.

item	specification
model	10B trainable parameters
GPUs	four gpuA40x4 nodes: sixteen A40s, 44 GB each
inside a node	50 GB/s GPU-GPU over PCIe
between nodes	10 GB/s GPU-GPU over RDMA
activations	reserve about 14 GB per GPU at micro-batch 1

Choose dp, tp and pp to use all 16 GPUs to train this model as fast as you can. Show that your choice fits in memory and explain why you did not shard further. Say which of your axes stay inside a node and which cross between nodes.

Choosing dp, tp and pp

1. How many bytes does each parameter need for its weight, gradient and Adam state? What is the total for 10B?

4 bytes per weight + 4 bytes per gradient + 4 x 2 bytes per Adam moment = 16 bytes

16 × 10B = 160 GB.

2. After 14 GB is reserved for activations how many ways must the state be split?

44 GB - 14 GB = 30 GB left. 160 GB ÷ 30 GB per GPU ≈ 5.3 GPUs, so at least 6 ways.

3. Keep tp inside a node (tp = 4). Fill in the table: what is the smallest split that fits?

4 × 2: 34 GB fits in 44 GB and leaves dp = 2. 4 × 4 also fits, but it splits further than memory needs.

Try a split

tp × pp	state per GPU	with 14 GB of activations	fits in 44 GB?	dp
4 × 1	**40 GB**	**54 GB**	**no**	**4**
4 × 2	**20 GB**	**34 GB**	**yes**	**2**
4 × 4	**10 GB**	**24 GB**	**yes**	**1**

Choosing which run to ship

GPT-2 medium (406M parameters) runs the screenwriting autocomplete. You adapted the same checkpoint to write restaurant-listing sentences, by full fine-tuning and by LoRA at four ranks.

Dev loss: lower is better. **BLEU:** higher is better. **FineWeb:** lower is better.

run	trainable	dev loss	BLEU	FineWeb
GPT-2 medium, no fine-tuning	0	n/a	9.3	3.016
full fine-tuning	406.3M	0.998	63.5	3.924
LoRA r = 1	0.39M	1.126	61.6	3.176
LoRA r = 4	1.57M	1.091	61.6	3.140
LoRA r = 16	6.29M	1.056	61.1	3.168
LoRA r = 64	25.2M	1.023	62.8	3.176

(b) Product: full fine-tuning has the best dev loss and the best BLEU, so ship it for both products. What is your response? Name the run you would ship and the column that decides it.

Choosing which run to ship

GPT-2 medium, same checkpoint for both products. Dev loss and FineWeb: lower is better. BLEU: higher is better.

run	dev loss	BLEU	FineWeb
no fine-tuning	n/a	9.3	3.016
full fine-tuning	0.998	63.5	3.924
LoRA r = 1	1.126	61.6	3.176
LoRA r = 4	1.091	61.6	3.140
LoRA r = 16	1.056	61.1	3.168
LoRA r = 64	1.023	62.8	3.176

1. The same checkpoint runs the autocomplete. What does FineWeb measure, and why should it decide?

Forgetting of general web text, the autocomplete's job. Full fine-tuning is about 0.9 nats worse; LoRA runs 0.12 to 0.16.

2. Why does full fine-tuning forget so much more than LoRA?

It updates every weight. LoRA freezes the base and learns a small low-rank change, so it learns less and forgets less.

3. Full fine-tuning gains 0.7 BLEU over LoRA r = 64. How do you tell whether that gain is real?

Compare it with the spread across LoRA ranks: 61.1 to 62.8 with no trend. 0.7 is within that noise.

4. Which run would you ship, and how would you serve it?

LoRA r = 1 or 4. Keep the adapter separate so the autocomplete runs the untouched base weights.

Updating the block table

You served GPT-2 XL in bf16 in Assignment 6 with a paged KV cache: keys and values live in one pool of fixed-size blocks, and a sequence gets a block only when it needs one.

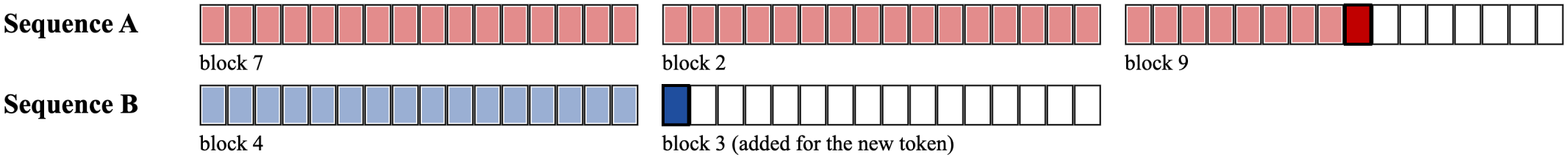
Assume a block size of 16. Sequence A has 40 tokens cached, with block table [7, 2, 9]. Sequence B has 16 tokens cached, with block table [4]. The engine runs one decode step for both, A in row 0 and B in row 1, and the allocator's next free physical block is 3.

(i) What is the block table for Sequence A and B after the decode step?
(ii) What is the flat slot (block ID and offset) of the new token this step writes for each sequence?
(iii) B's row is padded with block 0 to match A's table length, but another running request owns block 0. Can B attend to the slots in block 0 (or the other request attend to B), and if not, why not?

Updating the block table

Blocks hold 16 tokens. Sequence A has 40 tokens cached, block table [7, 2, 9]. Sequence B has 16 cached, block table [4]. The next free block is 3.

The slots in each sequence's blocks. Red: Sequence A. Blue: Sequence B. Solid with a heavy outline: the new token written this decode step.



1. Does each block table have room for one more token? If not, add the next free block.

A: 48 slots, needs 41, stays [7, 2, 9]. B: 16 slots, needs 17, becomes [4, 3].

2. The position is the number already cached. Divide by 16: quotient = entry, remainder = offset.

A's new token is at position 40, and 40 ÷ 16 = 2 r 8, so it goes in entry 2 (block 9) at offset 8. B's is at position 16, and 16 ÷ 16 = 1 r 0, so it goes in entry 1 (block 3) at offset 0.

3. B's row is padded to [4, 3, 0]. Which slots may its new token see ($s \leq p$)? Is block 0 among them?

Slots 0 to 16. Block 0 (slots 32 to 47) is masked, so no.

Section 2 of 3

Medium Difficulty Problems

Comparing base and the twin

Two models identical in architecture and budget (10.6M non-embedding parameters, about 82M characters seen). Base trained on the 1.0M-character Shakespeare split for about 80 passes; the twin trained on 82M characters of TinyStories. Loss is nats per character on held-out text.

model	shakespeare	tinystories
base	1.43	2.13
twin	2.62	0.70

(b) A teammate reads the twin's 0.70 against base's 1.43 and concludes the twin is the better model. What is your response? Say also why the twin's Shakespeare number is so much worse than base's TinyStories number, when the two models are identical in size and budget.

Comparing base and the twin

1. Which comparisons in this grid keep the exam the same?

Down a column. 0.70 and 1.43 are on different exams.

2. Why is the twin worse on Shakespeare than base is on TinyStories? Think about what each training set contains.

TinyStories' words mostly appear in Shakespeare, not the reverse. The twin's narrow training also makes it sharp, so it pays more on unfamiliar text.

Loss in nats per character (lower is better)

	Shakespeare exam	TinyStories exam
base	1.43	2.13
twin	2.62	0.70

Reading the pairwise judge results

Gemini compares a base sample with a twin sample under the TinyStories rubric, 50 pairs. Every pair is judged twice with A and B swapped. A pair is a win only if the same model wins both times; if the two verdicts disagree, the pair is flipped and scored as a tie.

base wins both	twin wins both	tie both	verdicts disagree (flipped)	pairs
10	4	5	31	50

Win rate over the 14 pairs with a consistent winner: base 10/14 = 0.71 ± 0.24 (95% interval). Claim (iii): base beats the twin on the TinyStories rubric with a 71% win rate.

(a) What does the flip count tell you about this judge run, and what do you check before the 71% goes in the report?

Reading the pairwise judge results

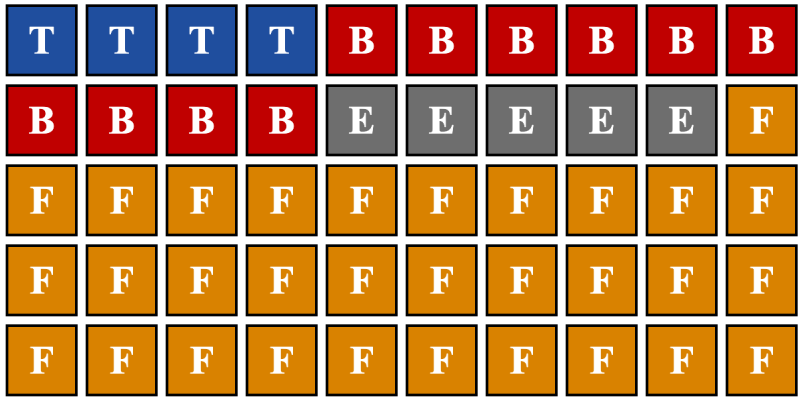
1. Only 19 of 50 pairs (38%) got the same verdict twice. How often would a judge picking at random be consistent?

About a third. 38% is close to random.

2. The twin should win under this rubric, but base came out ahead. How would you check whether the judge result is real?

Check judge_model in the results JSON (does it say mock?), read a few traces in Opik, then run --limit 5 against the real judge.

One square per judged pair. Each press fills one group: T (blue) = twin wins both times (4), B (red) = base wins both times (10), E (grey) = tie both times (5), F (orange) = verdicts disagree (31).



What exact dedup removes

Your Assignment 3 Spark pipeline turns five raw WET files (116,771 pages) into a clean, deduplicated corpus in six stages. You then train at a fixed 50M-token budget.

(a) Which stage removes the most data, and what do the rows column and the MB column together tell you about the lines it removed? Say why that matters at a fixed 50M-token budget.

stage	rows	% of lines	MB	seconds
1 ingest	116,771	n/a	1073.2	44.1
2 chunk	22,461,175	100.00%	1050.2	26.6
3 normalize	19,024,390	84.70%	684.6	67.8
4 features	19,024,390	84.70%	684.6	70.0
5 exact dedup	6,093,868	27.13%	437.6	40.9
6 write	6,093,868	27.13%	437.6	15.3

What exact dedup removes

From the pipeline table:

stage	rows	MB
into stage 5 (after 4 features)	19,024,390	684.6
out of stage 5 (exact dedup)	6,093,868	437.6

1. Which stage removes the most rows? What fraction does it remove?

Exact dedup: 12.9M of 19.0M rows, about 68%.

2. What fraction of the bytes does it remove?

247.0 of 684.6 MB, about 36%.

3. Based on these numbers, what do you expect the deduplicated data is?

Short, repeated boilerplate: removed lines average about 19 bytes, kept lines about 72.

4. Training uses a fixed 50M-token budget. Why does it matter that these lines are gone?

With a fixed budget, repeated boilerplate would take tokens away from real text.

Naive and overlapped DDP

You train medium (406M parameters) with data parallelism on one node's four A40s, global batch 128. Each rank runs its 32 sequences as 8 micro-batches of 4, and no_sync() means gradients are reduced only during the eighth micro-batch.

strategy	ms/step
perfect scaling (one GPU's step ÷ 4)	1412
naive DDP	1595
overlapped DDP	1537

(a) Explain the benefit of overlapped DDP compared to naive DDP. Be specific about what exactly is being overlapped, why this is different from naive DDP, and why this would improve the performance compared to naive DDP.

Naive and overlapped DDP

Backward runs from the last layer (L4) to the first (L1). Each layer's gradients have to be all-reduced across GPUs before the optimizer step.

1. Naive DDP: when does the all-reduce start?

After the whole backward pass: one long all-reduce at the end while the GPUs wait.

2. Overlapped DDP: when are each layer's gradients ready, and when can its all-reduce start?

Right after each layer's backward. AR L4 runs under backward L3, and so on.

3. What is running while each all-reduce happens, and why does that make the step shorter?

The backward pass of earlier layers. Communication is hidden behind it: 58 ms saved.

The memory LoRA does not remove

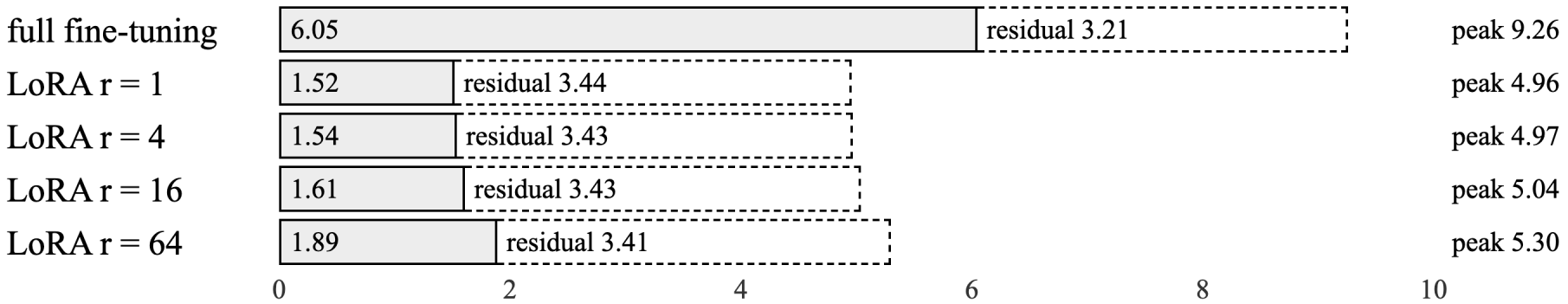
GPT-2 medium (406M parameters, 24 layers, width 1024, 50,257-token vocabulary) on one 24 GiB card. GiB: predicted = weights + gradients + optimizer state, counted from the parameters; peak = measured; residual = peak − predicted.

run	trainable	predicted	peak	residual
full fine-tuning	406.3M	6.05	9.26	3.21
LoRA r = 1	0.39M	1.52	4.96	3.44
LoRA r = 4	1.57M	1.54	4.97	3.44
LoRA r = 16	6.29M	1.61	5.04	3.43
LoRA r = 64	25.2M	1.89	5.30	3.41

(b) Every GPT-2 medium run's residual sits between 3.21 and 3.44 GiB, while trainable parameters range from 0.39M to 406M. What is the residual, and why does LoRA not shrink it?

The memory LoRA does not remove

GiB. Grey: predicted memory. Dashed: the residual, peak minus predicted.



1. Predicted memory counts weights, gradients and optimizer state. What else does training keep in memory?

Activations, plus fp32 logits over 50,257 tokens.

2. Which of those does LoRA remove, and which does it leave? Why?

It removes gradients and Adam state, but keeps activations, which are needed to backprop to the adapters.

3. What could you change to make the residual smaller?

Smaller batch, shorter sequences, gradient checkpointing.

Which positions carry loss

You adapt GPT-2 medium to write one sentence per restaurant listing (E2E). One training pair: attributes name[The Vaults], eatType[pub], priceRange[more than £30], customer rating[5 out of 5], near[Café Adriatic]; sentence “The Vaults pub near Café Adriatic has a 5 star rating. Prices start at £30.”

Each training example is one token sequence: a fixed instruction line and the attribute list (the prompt), then the sentence and an end-of-text token (the target). The loss is next-token cross-entropy, and the data loader marks which positions to leave out of it.

(a) A new hire’s collate leaves out padding only, so the prompt is scored along with the sentence. Which positions should carry loss, and why? What does the new hire’s version spend training on, and what does it do to the dev-loss number?

Which positions carry loss

One training example, left to right

segment	instruction line	attribute list	sentence	end-of-text	padding
example	[the fixed instruction]	name[The Vaults], eatType[pub], …	The Vaults pub near Café Adriatic has a 5 star rating. …	<|endoftext|>	pad pad
should carry loss?	**No**	**No**	**Yes**	**Yes**	**No**

1. At serving time, which segments does the booking site supply, and which must the model write?

The site supplies the instruction and attributes. The model writes the sentence and the end token.

2. Fill in the bottom row with yes or no.

3. What is the new hire's version spending its training on? Why can't its dev loss be compared with 0.998?

Predicting the fixed prompt text. Its dev loss averages over different tokens, so it is not comparable.

KV cache size per token

You served GPT-2 XL in bf16 in Assignment 6: 48 layers, a 1024-token context, hidden dimension 1600 (25 attention heads of 64), each head with its own keys and values. The KV cache keeps every layer's key and value for each token. A contiguous cache reserves a fixed run of slots for each request.

(a) How many bytes of KV cache does one token take? How many requests fit in a 1 GB (10^9 bytes) KV pool if the serving engine uses a contiguous cache? Show your work.

KV cache size per token

1. For one token in one layer, what does the cache store, and how many numbers is that?

K and V: 2 vectors
each: 25 heads × 64 =
1600 numbers

2. Multiply by the number of layers and the bytes per number. How many bytes per token?

2 (K, V)
× 48 layers
× 1600 numbers
× 2 bytes (bf16)
= 307,200 bytes per token

3. How many tokens does a contiguous cache reserve for each request, and why? How many bytes is that?

Length is unknown, so reserve all 1024 tokens
1024 × 307,200 bytes
≈ 314.6 MB per request

4. Divide the 1 GB pool by that, and mark the requests on the bar below.

$10^9 \div 314.6$ MB ≈ 3.2
so 3 requests fit
about 56 MB left over



Why the paged cache serves more tokens

The same GPT-2 XL server with a 1 GB KV pool. Part (a): 300 KiB per token, so a contiguous cache fits 3 requests.

(b) Decoding one step takes 39 ms with a contiguous cache and 76 ms with a paged one. Yet with a 1 GB pool, continuous batching serves 473 tokens/s with a paged cache, while a contiguous cache would serve only about 80 tokens/s. First, explain where the extra time went for the paged cache to decode a step. Then, using your answer to part (a), explain why the paged cache is still advantageous despite its slower steps.

Why the paged cache serves more tokens

1. What extra work does the paged cache do in each step?

Copying each sequence's blocks into one contiguous, padded tensor: the extra 37 ms.

2. From part (a): how many tokens per second for contiguous?

3 requests × 1000 ÷ 39 ≈ 77 tokens/s.

3. Why do more requests fit with a paged cache?

Blocks are given only as needed, so no request reserves the full 1024-token context.

4. Put it together using tokens per step ÷ time per step.

About 2 times slower per step, but about 12 times the tokens per step (473 × 0.076 ≈ 36), so about 6 times the tokens/s.

What to do with an evicted request’s cache

Your Assignment 6 serving engine. You also implemented an optimistic admission policy that only budgets memory needed for the next step, instead of reserving memory for the entire request. The engine must evict a request if it ends up running out of memory.

(b) Discuss two options for what the engine can do with the evicted request’s KV cache, and the trade-off between them.

What to do with an evicted request's cache

1. Name two things the engine can do with an evicted request's cache.

Recompute: discard the cache. Swap: copy it to CPU memory.

2. For each, what happens when the request comes back?

Recompute: prefill the prompt and generated tokens again. Swap: copy the blocks back to the GPU.

3. What does each option cost, and when would you prefer one over the other?

Recompute costs GPU compute. Swap costs CPU memory and PCIe transfers, but no extra compute.

Section 3 of 3

Easier Questions

The attention bug

Your Assignment 1 model, a character-level decoder-only transformer with a 256-character context and the 65-character vocabulary that data/prepare.py built from the 1.1M-character Shakespeare corpus. Before training, it had to pass four tests: test_output_shapes, test_causality, test_init_loss_near_uniform and test_can_overfit_tiny_batch.

A teammate rewrites CausalSelfAttention. Their version passes the shape test and the overfit test, and its validation loss falls below 0.3 nats per character within 200 iterations, well under the 1.46 the reference run reaches after 5,000.

(a) Name the most likely bug. Explain why it produces a number that good, in terms of which positions each position is allowed to attend to.

The attention bug

1. Write the target for each position. Which input position holds it?

2. Shade what a correct mask allows. Mark each position's target cell.

3. With no mask, what can each position do? What happens to the loss?

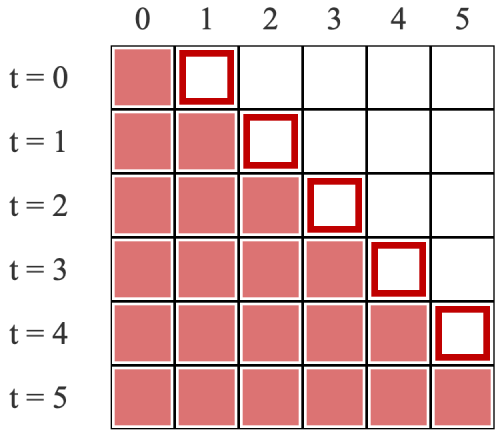
It sees t + 1 and copies its own answer, so the loss collapses toward zero.

4. What happens when this model generates new text?

There is no future to copy, so it generates garbage.

Each position predicts the next character.

position	0	1	2	3	4	5
input	R	O	M	E	O	:
target	**O**	**M**	**E**	**O**	**:**	**?**



rows: position t; columns: position s it may look at. Outlined: each position's target.

The line-level filters

Five mixes of your Assignment 3 web corpus, each at the same 50M-token budget; loss in nats per character. The last three rows all extend doc_stopwords >= 5 AND n_words >= 12.

mix	query	wikipedia
baseline	true	2.16
english_pages	doc_stopwords >= 5	2.04
ends_sent	… AND ends_sentence	1.79
stopdense	… AND stop_hits >= 3	1.99
middle	… AND rel_pos BETWEEN .1 AND .9	1.93

(b) Your teammate is confused about why the queries in the last three mixes would improve performance on the Wikipedia dataset. Explain the rationale behind ends_sentence, stop_hits >= 3 and rel_pos as features: what does each query do, and why is it a signal for better Wikipedia performance?

The line-level filters

Write keep or drop for each line under each filter. Each press fills in one row.

line of a web page (made-up example)	rel_pos	ends_sentence	stop_hits ≥ 3	rel_pos .1 to .9
Home - News - Sports - Login	0.00	**drop**	**drop**	**drop**
A flood came to the town in the night and took much of it.	0.20	**keep**	**keep**	**keep**
Click here to read more	0.40	**drop**	**drop**	**keep**
Price: $33. Ships free.	0.60	**keep**	**drop**	**keep**
Many of the homes had to be rebuilt in a year and a day.	0.80	**keep**	**keep**	**keep**
Example News. All rights reserved.	1.00	**keep**	**drop**	**drop**

1. Fill in keep or drop for each line under each filter.

Only the two prose lines (the flood and the homes) pass all three filters.

2. In a sentence each, what kind of line does each filter remove?

Fragments and menus; lines that aren't prose; headers, navigation and footers.

3. Why would removing those lines help the model on Wikipedia text?

What remains looks like Wikipedia: complete English sentences.

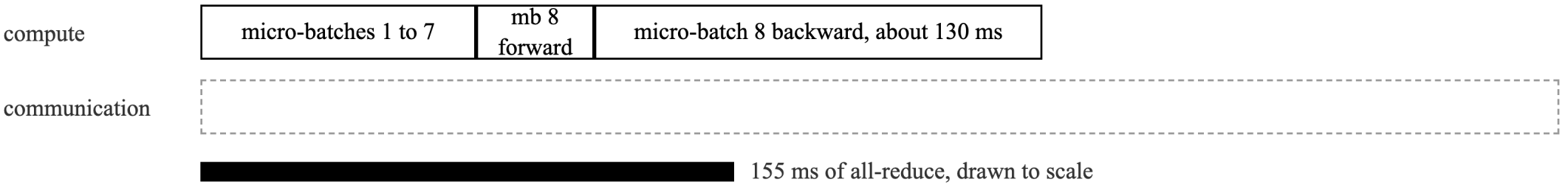
How much communication overlap can hide

You train medium (406M parameters) with data parallelism on one node's four A40s, global batch 128. Each rank runs its 32 sequences as 8 micro-batches of 4, and no_sync() means gradients are reduced only during the eighth micro-batch.

strategy	ms/step
perfect scaling (one GPU's step ÷ 4)	1412
naive DDP	1595
overlapped DDP	1537

(b) Overlap brings the step from 1595 to 1537 ms. However, from your all-reduce profiling you expect the communication to take 155 ms. Why does overlapping only hide 58 of the 155 ms, and not all of it?

How much communication overlap can hide



One training step on one rank. The first box is not to scale.

1. How much time did overlap save? How much of the 155 ms is still exposed?

58 ms saved (1595 − 1537). 97 ms still exposed.

2. With no_sync, which backward pass can the all-reduce overlap with, and how long is it?

Only micro-batch 8's backward, about 130 ms, which is shorter than 155 ms.

3. Which gradients are computed last, and what computation is left to overlap with them?

The first block and the token embedding. No computation is left after them, so that part is exposed.

AFAB and 1F1B schedules

You train large cut into four pipeline stages on one node (pp = 4), global batch 32, split into m micro-batches. A teammate switches the schedule from AFAB (all forward, all backward) to 1F1B and records:

m	sequences per micro-batch	AFAB ms/step	AFAB peak GiB	1F1B ms/step	1F1B peak GiB
4	8	1134	17.12	1132	17.12
8	4	1015	18.42	1013	11.55
16	2	1017	21.12	1015	8.36
32	1	out of memory	out of memory	1197	6.71

Explain the results of the switch from AFAB to 1F1B. Specifically, what is the difference in the schedule between AFAB and 1F1B, and why does this lead to the observed differences in metrics?

AFAB and 1F1B schedules

Look at stage 0 of a 4-stage pipeline (p = 4) running m = 8 micro-batches. A micro-batch is "held" from the moment its forward finishes until its backward runs, because its activations stay in memory the whole time.

1. AFAB runs F1 to F8, then B1 to B8. How many micro-batches does stage 0 hold at its peak?

All 8. Every forward runs before any backward, so the count climbs from 1 to 8 before it starts to fall. The peak equals m.

2. 1F1B warms up with p − 1 − i forwards, then alternates one forward and one backward. What is the order for stage 0, and what is its peak?

F1 F2 F3 F4 B1 F5 B2 F6 B3 F7 B4 F8 B5 B6 B7 B8. After the warm-up, every new forward is paired with a backward, so stage 0 never holds more than 4. The peak equals p.

3. Use the two peaks to explain the memory column. Why does the time column barely change?

AFAB holds m and runs out at m = 32; 1F1B holds at most p. Same work and bubble, so similar time.

Fitting GPT-2 XL on one card

A4’s trainer stores every weight in fp32, computes in bf16, and uses Adam. Each card has 24 GiB. Full fine-tuning updates every weight; LoRA freezes every original weight and trains a small pair of low-rank matrices beside each of the four linear layers in every block.

GPT-2 medium run	trainable	predicted GiB	peak GiB
full fine-tuning	406.3M	6.05	9.26
LoRA r = 4	1.57M	1.54	4.97

(a) A second customer needs a bigger model, and product proposes full fine-tuning GPT-2 XL (1.64B parameters, 48 layers, width 1600) on one of the 24 GiB cards. Will it fit? What about LoRA at rank 4? Count the bytes each weight needs.

Fitting GPT-2 XL on one card

1. What is stored for a trainable weight, and for a frozen one? How many bytes each?

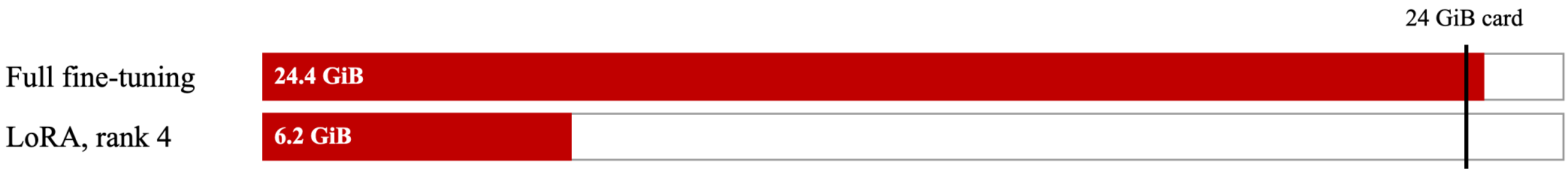
Trainable: 4 (weight) + 4 (gradient) + 8 (two Adam moments) = 16 bytes. Frozen: 4 bytes, the weight only.

2. Full fine-tuning: multiply by 1.64B parameters. Does it fit before any activations?

16 bytes per parameter × 1.64B parameters ≈ 24.4 GiB of weights, gradients and Adam state. That is over the 24 GiB card before any activations, so it doesn't fit.

3. LoRA at rank 4: the frozen base plus the adapters. How many adapter parameters are there? Does it fit?

About 4.9M adapter parameters. Frozen base: 1.64B × 4 bytes ≈ 6.1 GiB. Adapters: 4.9M × 16 bytes ≈ 0.07 GiB. About 6.2 GiB in total, so it fits.



Each track runs from 0 to 26 GiB. The bars show each run's memory before activations.

Static and continuous batching

Your Assignment 6 serving engine with GPT-2 XL. A 4 GB pool serves 400 requests that arrive at once, with outputs of 8 to 128 tokens, at most 64 running at a time.

A static batching scheduler averages 31 requests per decode step and 610 tokens/s. A continuous batching scheduler averages 54 and 747 tokens/s.

(a) Explain what continuous batching is doing differently, and why it is better than static batching.

Static and continuous batching

At most 2 sequences run at once, so there are 2 slots. A and B start at step 1; C and D wait, in that order. Each cell shows which sequence a slot is generating at that step.

Sequence A: 2 tokens Sequence B: 6 tokens Sequence C: 2 tokens Sequence D: 3 tokens finished

Static batching

step	1	2	3	4	5	6	7	8	9
Slot 1	A	A	idle	idle	idle	idle	C	C	idle
Slot 2	B	B	B	B	B	B	D	D	D

Continuous batching

step	1	2	3	4	5	6	7	8	9
Slot 1	A	A	C	C	D	D	D		
Slot 2	B	B	B	B	B	B	idle		

1. Static: C and D can start only when A and B have both finished. Which sequence runs in each slot at each step?

A finishes after step 2, but its slot sits idle until B finishes at step 6. Only then do C and D start, so the batch takes 9 steps.

2. Continuous: a waiting sequence takes a slot in the step after it frees up. Which sequence runs where?

C moves into A's slot at step 3, and D moves in as soon as C finishes. No slot waits on B, so everything is done after 7 steps.

3. Count the busy slots in each grid. How does this explain 31 versus 54 requests per step in the quiz?

Static uses 13 of 18 slot-steps (72%); continuous uses 13 of 14 (93%). Refilling slots right away keeps more requests running, which is why the quiz saw 54 per step instead of 31.