Outcome Prediction Quiz: Capacity

CS 312 · 40 minutes · 90 points

The final answer submission is now four pages instead of the usual two.

How to answer. This quiz has six questions. Predict the outcomes using your A5 experiments. Put your name, SUNet ID, choices and explanations on the four answer pages S1–S4 at the end. Print them double-sided on two sheets; only those pages are graded. Evidence and reasoning can recover up to 75% of missed points.

Reference recipe. Use A5 defaults: d8 ($D = 8$, $W = 512$), vocabulary 4096, context 1024, 600,000 training sequences (614.4M tokens), batch 64, LR .003 and seed 42. Loss comparisons use the full default validation set; lower is better. Router diagnostics follow A5's eight-sequence, 128-position example. All LRs below are peak values: use 1% warmup followed by linear decay. Each part states its peak LR or tuning procedure. LR searches keep seed 42, data order and the full token budget fixed, and select by final validation loss. Parameter counts include embeddings and the LM head.

Q1. Loss comparisons

Difficulty: Easy

12 points

Code changes from A5: Appendix A.1, page 11.

Use the depth/width comparison from A5 Problem 1. "Normal" means $W/D = 64$; multipliers change this ratio at roughly the d8 parameter budget (34.6M).

Lightning round (12 points; 3 per row).

TLDR: Effect of width and depth on loss at a fixed parameter budget.

Write $<$, $=$ or $>$ in each box to compare the models' validation losses; lower loss is better. Train every shape from scratch with peak LR .003, using the reference recipe. Treat gaps below .02 NLL as ties.

Model	D	W	W/D	Parameters (M)
Super wide	1	1280	1280.0	34.247
8× normal ratio	2	960	480.0	34.596
2× normal ratio	5	640	128.0	34.947
Normal	**8**	**512**	**64.0**	**34.613**
$\frac{1}{2}$× normal ratio	13	416	32.0	36.041
$\frac{1}{8}$× normal ratio	34	256	7.5	34.428
Super narrow	62	192	3.1	34.746

	Left model's loss		Right model's loss
(a)	Super narrow	☐	Super wide
(b)	$\frac{1}{2}$× normal aspect ratio	☐	2× normal aspect ratio
(c)	$\frac{1}{8}$× normal aspect ratio	☐	8× normal aspect ratio
(d)	Super narrow	☐	Normal

Shapes are rounded to match the parameter budget; actual ratios and counts are shown above.

Record your choice, assignment evidence and rationale on the separate answer sheet.

Q2. Looping

Difficulty: Easy

20 points

Code changes from A5: Appendix A.2, page 11.

(a) More passes at inference (10 points).

TLDR: Effect of inference pass count on validation loss.

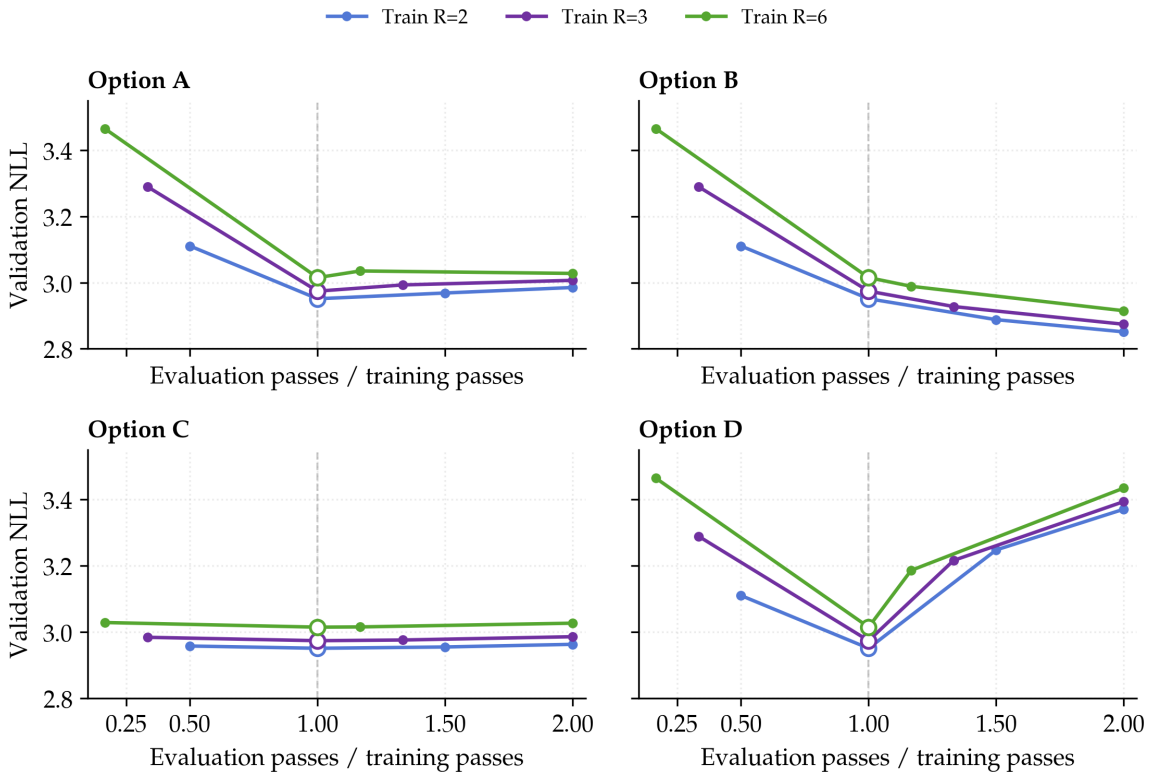
Use A5's prelude–shared core–coda models, each executing eight layers during training:

Model	Core blocks	Training passes R
Two-pass	3	2
Three-pass	2	3
Six-pass	1	6

Share all core parameters. After every pass, including the last, apply $h \leftarrow \text{RMSNorm}(h) + h_{\text{prelude}}$ with non-affine RMSNorm.

Use peak LR .003 for all three models. Keep each trained checkpoint fixed while varying inference passes.

Evaluate $m \in \{1, R, R+1, 2R\}$ passes on the same held-out inputs without changing weights or the boundary map. Which plot best shows loss? The x-axis is evaluation passes divided by training passes, m/R.



Choice: A ☐ B ☐ C ☐ D ☐

Record your choice, assignment evidence and rationale on the separate answer sheet.

(b) Growing a looped model (10 points).

TLDR: Effect of unsharing a loop halfway through training on final loss.

Use the *plain* six-pass model (no extra boundary RMS or recall). All three plans use peak LR .003, and the reference token budget:

- **Loop:** keep the shared block throughout.
- **Grow:** halfway through training, clone the shared block into six independently trainable copies and continue.
- **Dense:** train standard d8 with independent blocks throughout.

All reset Adam halfway and keep the data and LR clocks. Growth leaves the other weights unchanged. All execute eight layers throughout. Given $L_{\text{dense}} < L_{\text{loop}}$ and $M = (L_{\text{dense}} + L_{\text{loop}})/2$, where does L_{grow} fall?

- **A.** $L_{\text{grow}} \geq L_{\text{loop}}$: as bad as or worse than keeping the loop.
- **B.** $M \leq L_{\text{grow}} < L_{\text{loop}}$: between the baselines, closer to Loop (or exactly midway).
- **C.** $L_{\text{dense}} \leq L_{\text{grow}} < M$: between the baselines, closer to Dense.
- **D.** $L_{\text{grow}} < L_{\text{dense}}$: better than training dense throughout.

Record your choice, assignment evidence and rationale on the separate answer sheet.

Q3. Low-rank bottleneck

Difficulty: Medium

16 points

Code changes from A5: Appendix A.3, page 12.

Use A5 Problem 2's Softmax, Sigsoftmax, MoS-4 and CPLM. Rank refers to the vocabulary head (per component for MoS); CPLM's copy branch stays unrestricted.

(a) Interventions on two wide models (6 points).

TLDR: Effect of readout interventions with and without a rank-64 bottleneck.

Compare two versions of Q1's **super wide** $D = 1$**,** $W = 1280$ model:

- **Wide:** the full-rank vocabulary head ($r = 1280$).
- **Wide + rank-64 bottleneck:** the same backbone with an induced rank-64 vocabulary-head bottleneck.

Use peak LR .003 for all methods; extra head parameters are allowed. For each model, compare Sigsoftmax, MoS-4 and CPLM to **its own Softmax baseline**. Write $<$, $=$ or $>$ in each box to compare the method's validation loss (left) with its same-rank Softmax loss (right). Gaps below .02 NLL count as equal (six comparisons, 1 point each).

Method	Rank 64			Rank 1280		
Sigsoftmax	L_{method}	☐	L_{Softmax}	L_{method}	☐	L_{Softmax}
MoS-4	L_{method}	☐	L_{Softmax}	L_{method}	☐	L_{Softmax}
CPLM	L_{method}	☐	L_{Softmax}	L_{method}	☐	L_{Softmax}

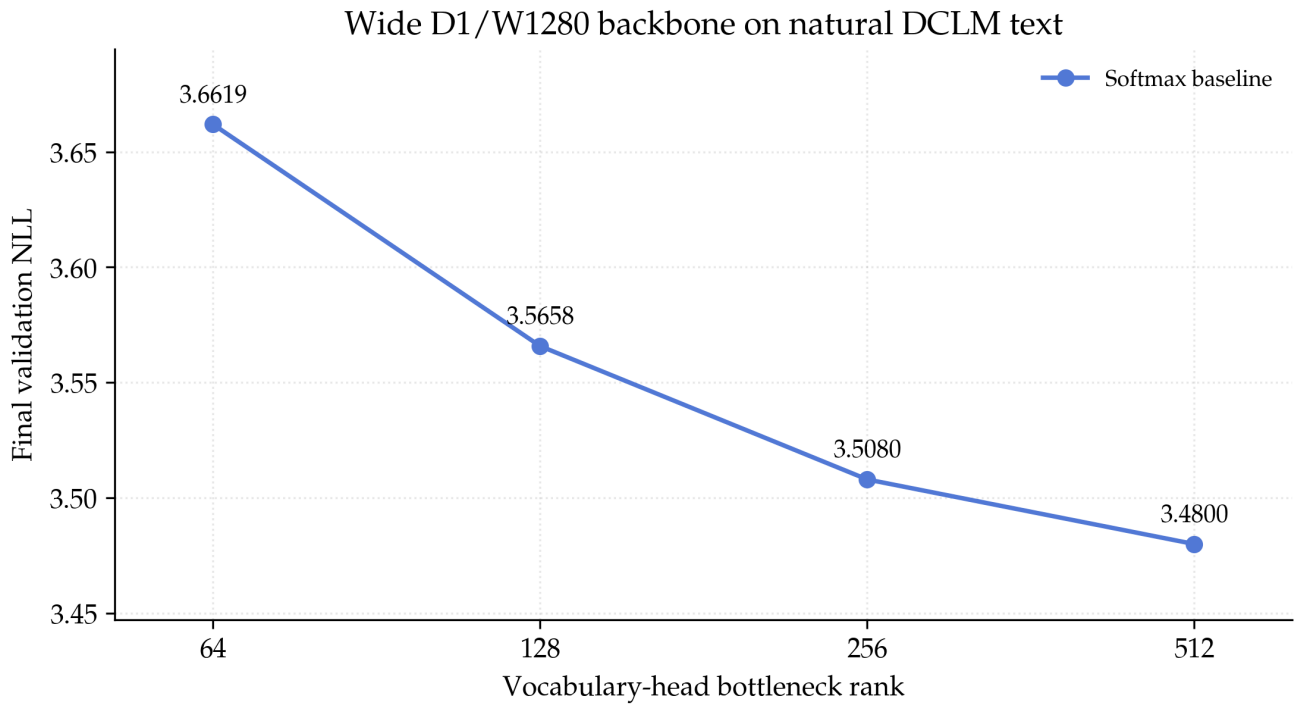
Record your choice, assignment evidence and rationale on the separate answer sheet.

(b) Matching the bottleneck rank (10 points).

TLDR: Loss of rank-64 MoS-4 relative to the Softmax rank ladder.

Use the same **wide** $D = 1$, $W = 1280$ **backbone on natural DCLM text**. The plot shows Softmax heads with bottleneck ranks 64, 128, 256 and 512; the backbone architecture stays fixed. Train every model from scratch at peak LR .003, seed 42 and the full reference token budget, measuring final loss on the full validation set.

Replace the rank-64 Softmax head with **MoS-4, rank 64 per component**. Which Softmax bottleneck rank has the **closest final validation loss** to this model?



- **A.** Rank 64.
- **B.** Rank 128.
- **C.** Rank 256.
- **D.** Rank 512.

Record your choice, assignment evidence and rationale on the separate answer sheet.

Q4. MoE routing methods

Difficulty: Medium

18 points

Code changes from A5: Appendix A.4, page 12.

Use A5 Problem 3's d8 MoE: four full-size experts per MLP, top-1 routing and $4p_{e^*}$ combine weights, except where stated. This stores more parameters than dense d8. Treat each package as a separate recipe. Letters A–D are defined independently in (a) and (b).

(a) Basic router packages (8 points).

TLDR: Validation-loss comparison of the basic router packages.

Use peak LR .003 for all four.

Package	Intervention
A	Random top-1: independent uniform routing per token/application; unit combine weight.
B	Frozen top-1: hold a random router matrix fixed; train the backbone and experts.
C	Learned top-1: train the router, without a balancing loss.
D	Dense: standard d8.

Write the **full ranking of final validation losses**, using each package letter once and the symbols $<$, $>$ and $=$. Here $<$ means lower loss, $>$ means higher loss, and $=$ means a tie (gap below .01 NLL).

Ranking: ☐ ☐ ☐ ☐ ☐ ☐ ☐

Record your choice, assignment evidence and rationale on the separate answer sheet.

(b) Improving the router (10 points).

TLDR: Validation-loss comparison of the router interventions.

Choose the package with the lowest final validation loss. Each package starts from the standard learned top-1 router and applies only its own listed intervention. All four use top-1 dispatch. A–C keep the standard combine weights; D replaces $4p_{e^*}$ by 1 in training and evaluation. For every package, test peak LRs $\{.001, .002, .003, .0045, .006, .009, .0135\}$. Use each package's lowest final validation loss.

Short English descriptions of these algorithms appear beside the corresponding diff in Appendix A.4 (page 12).

Package	Intervention
A	Cloned expert initialization
B	Auxiliary load balancing ($\lambda = .01$)
C	Quantile load balancing
D	Remove probability combine weight (use weight 1)

Select only one choice: A ☐ B ☐ C ☐ D ☐

Credit: best/second/third/fourth = 10/3/3/0 points.

Record your choice, assignment evidence and rationale on the separate answer sheet.

Q5. Router metrics across layers

Difficulty: Hard

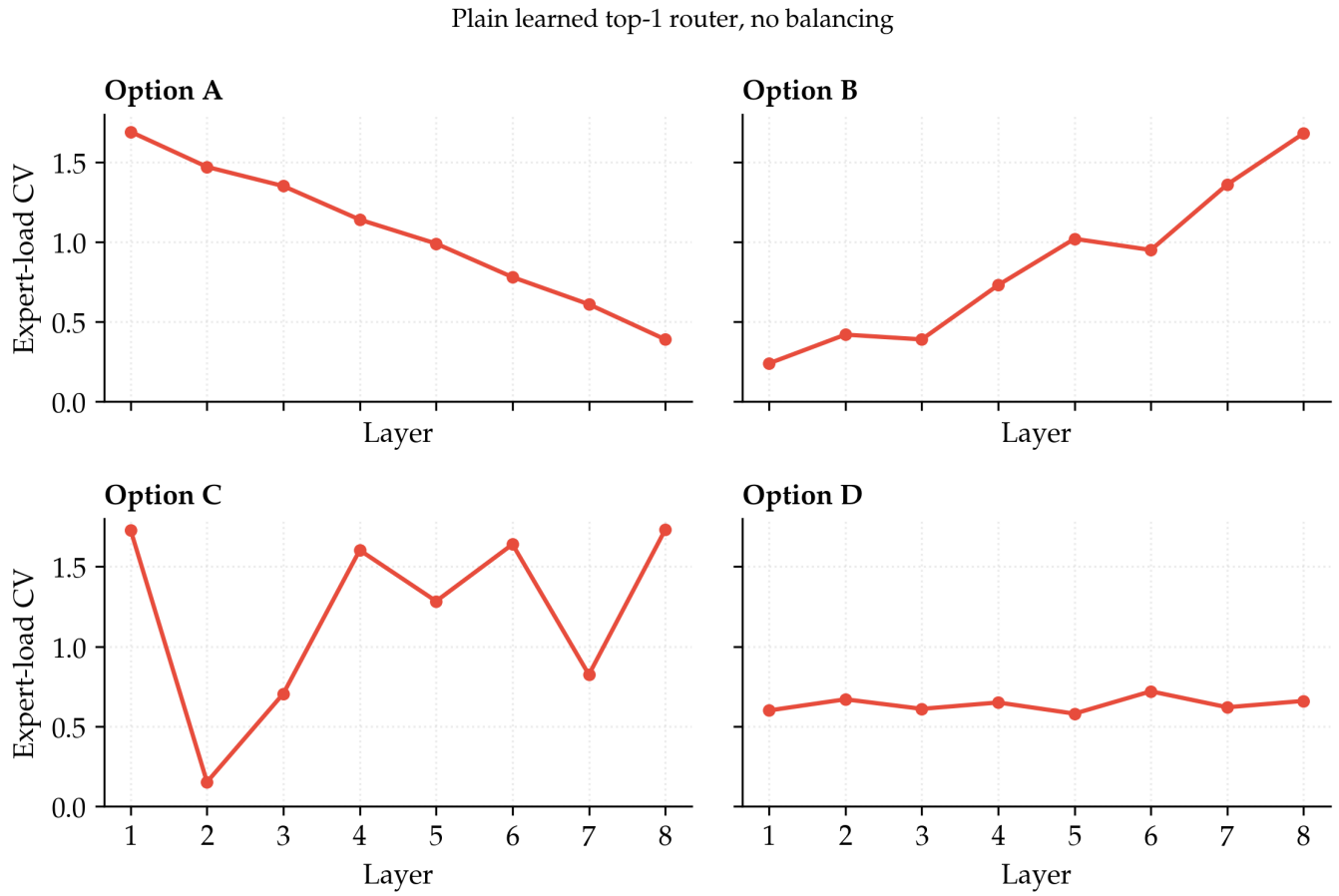
16 points

Code changes from A5: Appendix A.5, page 13.

(a) Expert-load CV across layers (8 points).

TLDR: Variation in expert-load CV across layers.

Use a **plain learned top-1 router with no balancing**, native $4p_{e^*}$ combine weights and peak LR .006, measured on A5's first eight validation sequences and first 128 positions. For selected expert counts $n_{\ell,e}$, use $\mathrm{CV}_\ell = \mathrm{std}_e(n_{\ell,e}) / \mathrm{mean}_e(n_{\ell,e})$ with population standard deviation. Which plot best shows CV_ℓ across layers?



Choice: A ☐ B ☐ C ☐ D ☐

Record your choice, assignment evidence and rationale on the separate answer sheet.

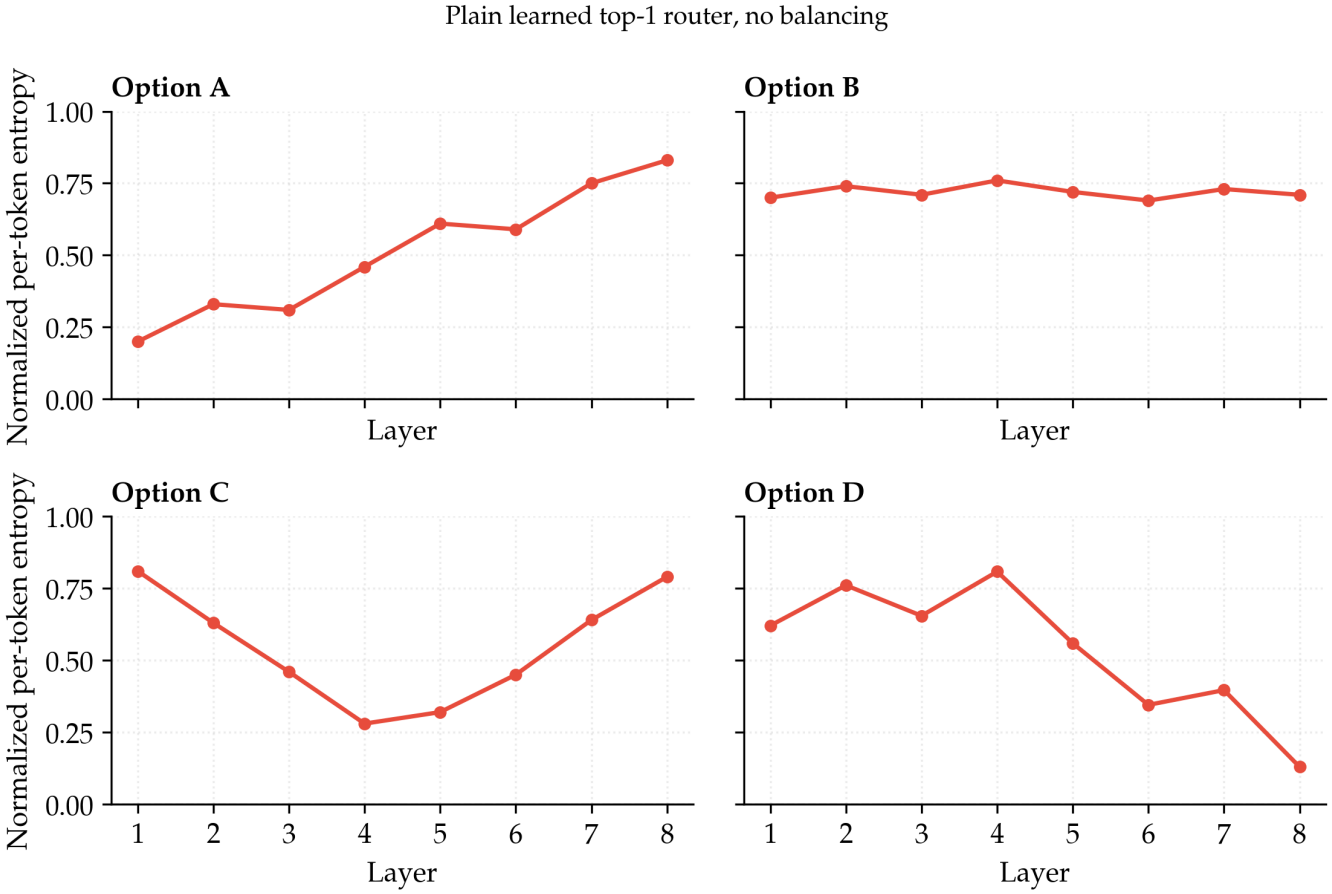
(b) Router entropy across layers (8 points).

TLDR: Variation in router entropy across layers.

Reuse (a)'s checkpoint trained at peak LR .006 and the same held-out inputs, with no further training or LR selection. At each layer, average the per-token entropy across all four router probabilities:

$$\overline{H}_\ell = -\frac{1}{T \log 4} \sum_{t=1}^{T} \sum_{e=1}^{4} p_{\ell,t,e} \log p_{\ell,t,e}.$$

This is 1 for uniform probabilities and 0 for one-hot probabilities. Which plot best matches the trained model? Judge the broad depth pattern, rather than individual wiggles.



Choice: A ☐ B ☐ C ☐ D ☐

Record your choice, assignment evidence and rationale on the separate answer sheet.

Q6. Allocating capacity between loops and experts

Difficulty: Hard

8 points

Code changes from A5: Appendix A.6, page 13.

TLDR: Best recipe for fixed unique parameter count and inference FLOPs

Use at most **dense d8's unique parameter count and inference FLOPs**, ignoring the small routing overhead. Every recipe has width 512, a **dense prelude and coda**, and eight block applications. Repeat the middle core as shown, sharing all core parameters across passes.

Package	Passes	Core blocks	Experts per core block	Unique params (M)
A	6	1	6	29.37
B	3	2	3	30.42
C	2	3	2	31.47
D	1	6	1	34.61

We swept peak LR over $\{.001, .002, .003, .0045, .006, .009\}$ for each package, using seed 42 and the full reference token budget.

Which package achieves the **lowest selected validation loss**?

Choice: A ☐ B ☐ C ☐ D ☐

Record your choice, assignment evidence and rationale on the separate answer sheet.

Appendix A: code changes from the A5 release

Each diff applies independently to the released starter at commit d240711. New recipe functions use A5's existing model classes and trainer. The combined patch and complete clean A5 folder are included with the answer-code files.

A.1 — Loss comparisons

```
--- /dev/null
+++ b/experiments/a5_capacity/quiz_recipes.py
@@ -0,0 +1,9 @@
+"""Quiz recipes: small changes to the released A5 implementation."""
+from dataclasses import replace
+
+
+def q1_configs():
+    from .sweeps import config
+    shapes = {'normal': (8, 512), 'narrow': (62, 192), 'wide': (1, 1280),
+              'half': (13, 416), 'twice': (5, 640), 'eighth': (34, 256), 'eight': (2, 960)}
+    return {name: config('dense', .003, depth=d, width=w) for name, (d, w) in shapes.items()}
```

A.2 — Looping

```
--- /dev/null
+++ b/experiments/a5_capacity/quiz_recipes.py
@@ -0,0 +1,27 @@
+"""Quiz recipes: small changes to the released A5 implementation."""
+from dataclasses import replace
+
+
+def q2_configs():
+    from .sweeps import config
+    output = {}
+    for passes in (2, 3, 6):
+        output[f'passes-{passes}'] = config('dense', .003,
+            model_builder='experiments.a5_capacity.p4_looping:LoopLM',
+            model_builder_kwargs=dict(passes=passes, boundary_norm=True, recall=1.))
+    for name in ('loop-reset', 'loop-to-unique', 'unique-reset'):
+        output[name] = config(name, .003)
+    return output
+
+def loop_losses(model, tokens, loss):
+    """Pass the FULL default validation tokens and a batched token-NLL helper."""
+    before = model.eval_passes
+    model.eval()
+    try:
+        values = {}
+        for passes in sorted({1, model.passes, model.passes+1, 2*model.passes}):
+            model.eval_passes = passes
+            values[passes] = loss(model, tokens)
+        return values
+    finally:
+        model.eval_passes = before
```

A.3 — Low-rank bottleneck

```
--- /dev/null
+++ b/experiments/a5_capacity/quiz_recipes.py
@@ -0,0 +1,26 @@
+"""Quiz recipes: small changes to the released A5 implementation."""
+from dataclasses import replace
+
+
+def q3_configs():
+    """Fixed wide backbone on natural text: readout interventions and rank ladder."""
+    from .sweeps import config
+    output = {}
+    for rank in (64, 1280):
+        for method in ('softmax', 'sigsoftmax', 'mos', 'cplm'):
+            builder = 'CPLM' if method == 'cplm' else 'HeadLM'
+            kwargs = dict(rank=None if rank == 1280 else rank)
+            if method != 'cplm':
+                kwargs['method'] = method
+            output[f'wide-{method}-{rank}'] = config('dense', .003, depth=1, width=1280,
+                model_builder=f'experiments.a5_capacity.p2_readout:{builder}',
+                model_builder_kwargs=kwargs)
+    for rank in (128, 256, 512):
+        output[f'wide-softmax-{rank}'] = config('dense', .003, depth=1, width=1280,
+            model_builder='experiments.a5_capacity.p2_readout:HeadLM',
+            model_builder_kwargs=dict(method='softmax', rank=rank))
+    return output
+
+def closest_rank(softmax_losses, mos64_loss):
+    """Compare final validation losses; ties choose the smaller rank."""
+    return min(softmax_losses, key=lambda rank: (abs(softmax_losses[rank]-mos64_loss), rank))
```

A.4 — MoE routing methods

Q4(b): independent interventions. Each package starts from the standard learned top-1 router and changes only the item below.

- **A — cloned experts.** Copy one expert's initial weights to all four experts. They then train independently; their weights are not tied.
- **B — auxiliary load balancing.** Add a loss with coefficient .01 that couples each expert's token share to its mean routing probability, encouraging a more even allocation of tokens.
- **C — quantile load balancing.** After each optimizer step, use score quantiles pooled across its microbatches to set expert biases for the next step, targeting an equal token share. These biases affect dispatch only; combine weights still use the original router probabilities.
- **D — unit combine weight.** Keep the top-1 expert selection, but replace the selected expert's output multiplier $4p_{e^*}$ with 1 during training and evaluation.

```
--- /dev/null
+++ b/experiments/a5_capacity/quiz_recipes.py
@@ -0,0 +1,13 @@
+"""Quiz recipes: small changes to the released A5 implementation."""
+from dataclasses import replace
+
+
+def q4_configs():
+    """Independent recipes: reset A--D between parts; (b) changes one intervention."""
+    from .sweeps import config
+    output = {f'a-{letter}': config(name, .003) for letter, name in
+              zip('ABCD', ('moe-random', 'moe-frozen', 'moe-learned', 'dense'))}
+    for letter, name in zip('ABCD', ('moe-cloned', 'moe-balanced', 'moe-quantile', 'moe-unit')):
+        for lr in (.001, .002, .003, .0045, .006, .009, .0135):
+            output[f'b-{letter}-{lr:g}'] = config(name, lr)
+    return output
```

A.5 — Router metrics

```
--- /dev/null
+++ b/experiments/a5_capacity/quiz_recipes.py
@@ -0,0 +1,12 @@
+"""Quiz recipes: small changes to the released A5 implementation."""
+from dataclasses import replace
+
+
+def q5_configs():
+    from .sweeps import config
+    return {'red': config('moe-learned', .006)}
+
+def router_layers(model, tokens):
+    """A5's diagnostic example: first8 validation sequences, first128 positions."""
+    from .diagnostics import measure_model
+    return measure_model(model, tokens[:8, :128])['router_layers']
```

A.6 — Loops versus experts

```
--- a/experiments/a5_capacity/p3_moe.py
+++ b/experiments/a5_capacity/p3_moe.py
@@ -18,10 +18,10 @@

 class Experts(nn.Module):
-    def __init__(self, config):
+    def __init__(self, config, experts=4):
         super().__init__()
         d, i = config.hidden_size, config.intermediate_size
-        self.gate = nn.Parameter(torch.empty(4, d, i))
-        self.up = nn.Parameter(torch.empty(4, d, i))
-        self.down = nn.Parameter(torch.empty(4, i, d))
+        self.gate = nn.Parameter(torch.empty(experts, d, i))
+        self.up = nn.Parameter(torch.empty(experts, d, i))
+        self.down = nn.Parameter(torch.empty(experts, i, d))
         self.reset_parameters()

@@ -46,13 +46,14 @@
 class RouterMLP(nn.Module):
     def __init__(self, config, *, routing='learned', norm=False, noise=0., clone=False,
-                 unit=False, balance=0., quantile=False, top_k=1):
+                 unit=False, balance=0., quantile=False, top_k=1, experts=4):
         super().__init__()
-        self.experts = Experts(config)
-        self.router = nn.Linear(config.hidden_size, 4, bias=False)
+        self.count = experts
+        self.experts = Experts(config, experts)
+        self.router = nn.Linear(config.hidden_size, experts, bias=False)
         self.norm = LlamaRMSNorm(config.hidden_size, 1e-5) if norm else nn.Identity()
         self.routing, self.noise, self.clone = routing, noise, clone
         self.unit, self.balance, self.quantile, self.top_k = unit, balance, quantile, top_k
         if routing in ('random', 'frozen'):self.router.requires_grad_(False)
-        self.register_buffer('bias', torch.zeros(4))
+        self.register_buffer('bias', torch.zeros(experts))
         self.pending = []
         self.aux_loss = None
@@ -65,9 +66,9 @@
         p = logits.softmax(-1)
         k = self.top_k
-        selected = torch.randint(4, (len(x), k), device=x.device) if self.routing == 'random' else (logits+self.bias).
     topk(k, -1).indices
-        weights = torch.ones_like(selected, dtype=x.dtype) if self.unit or self.routing == 'random' else (4/k)*p.gather
     (-1, selected).to(x.dtype)
-        counts = torch.bincount(selected.flatten(), minlength=4)
+        selected = torch.randint(self.count, (len(x), k), device=x.device) if self.routing == 'random' else (logits+
     self.bias).topk(k, -1).indices
+        weights = torch.ones_like(selected, dtype=x.dtype) if self.unit or self.routing == 'random' else (self.count/k)
     *p.gather(-1, selected).to(x.dtype)
+        counts = torch.bincount(selected.flatten(), minlength=self.count)
         shares = counts.float()/selected.numel()
-        self.aux_loss = self.balance*4*(shares.detach()*p.mean(0)).sum()
+        self.aux_loss = self.balance*self.count*(shares.detach()*p.mean(0)).sum()
```

```
         if self.training and torch.is_grad_enabled() and self.quantile:
             alpha = (logits.detach()+self.bias).topk(k+1, -1).values[:, -1:]
@@ -78,10 +79,10 @@
         counts_cpu = counts.tolist()  # One routing synchronization per layer.
         capacity = max(counts_cpu)
-        if capacity <= 1.5*selected.numel()/4:
+        if capacity <= 1.5*selected.numel()/self.count:
             expert = selected.flatten()[order]
             offset = torch.repeat_interleave(counts.cumsum(0)-counts, counts)
             slot = expert*capacity + torch.arange(len(tokens), device=x.device)-offset
-            packed = x.new_zeros((4*capacity, x.shape[-1])).index_copy(0, slot, x[tokens])
-            values = self.experts.compute(packed.view(4, capacity, -1)).flatten(0, 1)[slot]
+            packed = x.new_zeros((self.count*capacity, x.shape[-1])).index_copy(0, slot, x[tokens])
+            values = self.experts.compute(packed.view(self.count, capacity, -1)).flatten(0, 1)[slot]
         else:
             chunks = x[tokens].split(counts_cpu)
@@ -98,5 +99,5 @@
             # Quantile threshold over all accumulation microbatches, for NEXT step.
             residual = torch.cat(self.pending)
-            target = max(1, len(residual)*self.top_k//4)
+            target = max(1, len(residual)*self.top_k//self.count)
             beta = residual.kthvalue(len(residual)-target+1, dim=0).values
             self.bias.copy_(-beta+beta.mean())
--- a/experiments/a5_capacity/p4_looping.py
+++ b/experiments/a5_capacity/p4_looping.py
@@ -132,2 +132,18 @@
     from .sweeps import launch_one
     launch_one('loop-r6', .003)  # Edit the variant and LR.
+
+
+class LoopMoELM(LoopLM):
+    """Plain shared core with E learned experts; dense prelude and coda."""
+    def __init__(self, config, *, experts=6, **kwargs):
+        from .p3_moe import RouterMLP
+        super().__init__(config, **kwargs)
+        for block in self.model.layers[1:-1]:
+            if not isinstance(block.mlp, RouterMLP):
+                block.mlp = RouterMLP(config, experts=experts)
+
+    def post_initialize(self):
+        from .p3_moe import RouterMLP
+        for module in self.modules():
+            if isinstance(module, RouterMLP):
+                module.experts.reset_parameters()
--- /dev/null
+++ b/experiments/a5_capacity/quiz_recipes.py
@@ -0,0 +1,14 @@
+"""Quiz recipes: small changes to the released A5 implementation."""
+from dataclasses import replace
+
+
+def q6_configs(lrs=(.001, .002, .003, .0045, .006, .009)):
+    from .sweeps import config
+    output = {}
+    for letter, passes in zip('ABCD', (6, 3, 2, 1)):
+        for lr in lrs:
+            extra = {} if passes == 1 else dict(
+                model_builder='experiments.a5_capacity.p4_looping:LoopMoELM',
+                model_builder_kwargs=dict(passes=passes, experts=passes))
+            output[f'{letter}-{lr:g}'] = config('dense', lr, **extra)
+    return output
```

Name _____ **SUNet ID** _____

your Stanford login, e.g. `jdoe`; not the 8-digit ID number

Write $<, =, >$ in comparison boxes; mark one choice elsewhere. Submit all four pages S1–S4.

Q1. Loss comparisons — 12 points

3 points per row	Left model's loss		Right model's loss
(a)	Super narrow	☐	Super wide
(b)	$\frac{1}{2}\times$ normal aspect ratio	☐	$2\times$ normal aspect ratio
(c)	$\frac{1}{8}\times$ normal aspect ratio	☐	$8\times$ normal aspect ratio
(d)	Super narrow	☐	Normal

Assignment problem cited	**Observations from the experiment**	**Your rationale**

Q2. Looping — 20 points

10 points per part	A	B	C	D
(a) Inference passes	☐	☐	☐	☐
(b) Growing a loop	☐	☐	☐	☐

Assignment problem cited	**Observations from the experiment**	**Your rationale**
(a)	(a)	(a)
(b)	(b)	(b)

Name _____ **SUNet ID** _____

your Stanford login, e.g. `jdoe`; not the 8-digit ID number

Q3. Low-rank bottleneck **16 points**

(a) 1 point per comparison; method loss on the left, same-rank Softmax loss on the right.

Method	Rank 64	Rank 1280
Sigsoftmax	L_{method} ☐ L_{Softmax}	L_{method} ☐ L_{Softmax}
MoS-4	L_{method} ☐ L_{Softmax}	L_{method} ☐ L_{Softmax}
CPLM	L_{method} ☐ L_{Softmax}	L_{method} ☐ L_{Softmax}

(b) Closest rank: 10 pts ☐ A:64 ☐ B:128 ☐ C:256 ☐ D:512

Assignment problem cited	**Observations from the experiment**	**Your rationale**
(a) Rank 64	(a) Rank 64	(a) Rank 64
(a) Rank 1280	(a) Rank 1280	(a) Rank 1280
(b) Rank matching	(b) Rank matching	(b) Rank matching

Name _____ **SUNet ID** _____

your Stanford login, e.g. `jdoe`; not the 8-digit ID number

Q4. MoE routing methods — 18 points

(a) Ranking (A–D): 8 pts

[ ] [ ] [ ] [ ] [ ] [ ] [ ]

Letters in large boxes; $<, >, =$ in small boxes.

(b) Lowest loss: 10 pts

Select only one choice:

☐A ☐B ☐C ☐D

Assignment problem cited	Observations from the experiment	Your rationale
(a)	(a)	(a)
(b)	(b)	(b)

Name _____ **SUNet ID** _____

your Stanford login, e.g. `jdoe`; not the 8-digit ID number

Q5. Router metrics across layers — 16 points

8 points per part	A	B	C	D
(a) Expert-load CV	☐	☐	☐	☐
(b) Router entropy	☐	☐	☐	☐

Assignment problem cited	**Observations from the experiment**	**Your rationale**
(a)	(a)	(a)
(b)	(b)	(b)

Q6. Loops versus experts — 8 points

Lowest loss after LR selection: ☐A ☐B ☐C ☐D

Assignment problem cited	**Observations from the experiment**	**Your rationale**