Quiz 3

601.471/671 NLP: Self-Supervised Models

Spring 2025

Johns Hopkins University

April 2025

Complete all questions.

Use additional paper if needed.

Time: 80 minutes.

Name of student: _____

Q1. Concepts, Foundations and Big Picture *13 points*

For each of the following questions, select the most appropriate answer.

1. (1 points) In which case is fine-tuning likely to be *more* effective than in-context learning?
   ○ When the task involves classifying short sentences into categories seen during pre-training
   ○ When the task requires generating creative stories based on a few keywords.
   ○ When the task involves processing documents longer than the model's context window
   ○ When you have only a few examples to demonstrate the task

   Solution: *3*

2. (1 points) What is self-consistency decoding in chain-of-thought prompting?
   ○ Sampling multiple reasoning chains and choosing the most frequent final answer.
   ○ Having multiple models discuss and agree on an answer.
   ○ Ensuring rephrased questions yield the same answer.
   ○ Fine-tuning the model until it gives consistent answers.

   Solution: *1*

3. (1 points) In RLHF training of a pre-trained language models (LM), what is the role of the *reward model*?
   ○ An LM fine-tuned on human feedback to produce safer or preferred responses.
   ○ A model trained on human preference data to score or rank the LM's outputs (given a prompt).
   ○ The original LM that generates initial responses before any human feedback is incorporated.
   ○ A rule-based filter that removes disallowed content from the LM's outputs based on human values.

   Solution: *2*

4. (2 points) Which are the elements of a typical RLHF pipeline to align LMs? Select all that apply.
   ☐ Collecting human preferences by comparing pairs of model outputs for the same prompt.
   ☐ Training a reward model to predict which output humans prefer.
   ☐ Fine-tuning the model using reinforcement learning guided by the reward model.
   ☐ Pre-training on unannotated text to learn general language patterns.
   ☐ Using hand-written rules to enforce specific behaviors.

   Solution: *1,2,3*

5. (1 points) What is the role of the KL penalty term added to the reward during policy-gradient?
   ○ Encourages exploration by diverging from the pre-trained model
   ○ Prevents reward hacking and preserves fluency
   ○ Stabilizes training by rescaling gradients in mixed-precision setups
   ○ Keeps the reward model aligned with the policy to avoid mode collapse

   Solution: *2*

6. (2 points) During inference-time scaling, Best-of-N rejection sampling is often used to improve answer quality. Which of the following must occur in a Best-of-N scheme? Select all that apply.
   ☐ The model generates N parallel chains (thoughts + answers) from its policy distribution.
   ☐ A reward or verifier function scores each generated answer before selection.
   ☐ The final answer is chosen purely by majority voting among the N answers.
   ☐ Beam search is used to generate each individual chain.
   ☐ The method trades extra inference cost for better outputs without updating model weights.

   Solution: *1,2,5*

7. (1 points) What problem do ZeRO and FSDP primarily solve in training large language models?
   ○ High memory usage, by sharding model states across GPUs instead of replicating them.
   ○ Redundant transformer layers, by pruning duplicates during training.
   ○ Slow inference, by parallelizing single-query computation across GPUs.
   ○ Manual tuning, by auto-adjusting hyperparameters without human input.

   Solution: *1*

8. (1 points) What is *knowledge distillation* in language model training?
   ○ Merging two trained models by averaging their parameters.
   ○ Compressing a model by removing layers or parameters.
   ○ Training a smaller *student* model to imitate a larger *teacher* model's outputs.
   ○ Using a low learning rate so the model slowly "distills" knowledge over time.

   Solution: *3*

9. (1 points) In model efficiency, what is the main advantage of quantizing a large language model?
   ○ Zeroing out some neurons to make the model sparser and faster.
   ○ Lower-precision numbers (e.g., 8-bit) for parameters to reduce memory and speed up inference.
   ○ Training without data by using quantized gradients.
   ○ Splitting the model into smaller sub-models run independently and recombined.

   Solution: *2*

10. (2 points) Which technique reduce memory usage during LM training? Select all that apply.
    ☐ Caching all activations in memory to avoid recomputation.
    ☐ Offloading model parameters or optimizer states to CPU or disk when not in use.
    ☐ Using FP16 instead of FP32 to reduce memory for weights and gradients.
    ☐ Using 64-bit precision for all computations to improve numerical stability.

    Solution: *2,3*

Q2. Language Models and Evaluating Them *10 points*

1. (3 points) Recall that an n-gram language model factorizes the probability of a sentence X to the sum of log probabilities of individual tokens $p(X) = \prod_t p(X_t \mid X_{t-1}, \ldots, X_{t-n})$, where each word X_i depends on the previous n tokens. However, a student has this novel idea of reversing the language model: Each word instead of depending on the **previous** n words, depends on the **following** n words. Under this construction, write the factorization of the probability of the sentence $p(X)$.

Solution:

$$p(X) = \prod_t p(X_t \mid X_{t+1}, \ldots, X_{t+n})$$

Your answer:

2. (3 points) Recall the definition of perplexity: $\text{ppl}(w_1, ..., w_n) = 2^H$ where $H = -\frac{1}{n}\sum_{i=1}^{n} \log_2 p(w_i|w_1, \ldots, w_i - 1)$. Assume that perplexity is measured over entirety of a large natural language corpus with n words (for a large n). Also assume that this corpus is made of V-many unique words.
(a) If the model always knows what word should follow a given context, what is the perplexity of the model? (b) Under what conditions the model achieves a perplexity of V? Show your argument.

Solution: *Answer: (a) 1, (b) the number of the words in the language and when get that when our LM predicts completely randomly.*

Your answer:

3. (4 points) Let's remember the Transformer architecture. Define the Self-Attention operation and compute its computational complexity when applied to a batch of b sequences, each with n tokens $\mathbf{x}_1, ..., \mathbf{x}_n$ where $\mathbf{x}_i \in \mathbb{R}^d$. Use matrix notation when providing your definition. In your notation, show the dimensions of the intermediate variables and the output of the attention block.

Solution: *Let the input tokens be $\mathbf{X} \in \mathbb{R}^{b \times n \times d}$, the key, query, and value are given by*

$$\mathbf{K} = \mathbf{X}\mathbf{W}^k, \mathbf{Q} = \mathbf{X}\mathbf{W}^q, \mathbf{V} = \mathbf{X}\mathbf{W}^v$$

Then the self-attention is given by

$$\mathbf{S} = softmax\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_q}}\right)\mathbf{V}$$

Regarding the dimensions, two things are important to show: (1) The dimensions are $\mathbf{Q}\mathbf{K}^\top$ are $b \times n \times n$ where n is the sequence length and b is the batch size. (2) The output dimension is the same as the output.

Your answer:

Q3. Adaptation via Prompting and Tuning *21 (+ 3 bonus) points*

1. Prabhav has a large pre-trained LM intended for deployment in multiple tasks (medical Q&A, legal classification, general chatbot).

1.1 (3 points) Explain briefly why directly fine-tuning the entire model separately for each of these tasks is expensive. Provide a simple mathematical example to illustrate your point.

Solution: *Directly fine-tuning the entire model separately for each task requires storing a full copy of the model for each task. For instance, with a 170 billion-parameter model (approximately 340GB per model), adapting to 3 tasks independently would require $3 \times 340GB = 1020GB$ of storage, rapidly becoming impractical as the number of tasks grows.*

Your answer:

1.2 (3 points) Consider a pre-trained language model parameterized by weight matrices $W \in \mathbb{R}^{d\times d}$. Prabhav decides to adapt this model using LoRA by introducing a low-rank update $\Delta W = BA$, where $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times d}$, and $r \ll d$. Clearly state the total number of trainable parameters introduced by a single LoRA update. Provide a numerical example when $d = 4096$ and $r = 8$.

Solution: *The number of trainable parameters for LoRA is $dr + rd = 2dr$. With $d = 4096$ and $r = 8$, we have $2 \times 4096 \times 8 = 65,536$ trainable parameters, significantly fewer compared to $d^2 = 4096^2 = 16,777,216$.*

Your answer:

1.3 (3 points) Suppose Prabhav increases the rank r of the LoRA update. How does this affect (i) expressiveness (the ability to capture complex task-specific adjustments), and (ii) computational cost per forward pass? Justify your answer clearly.

Solution: *(i) Increasing r enhances the expressiveness of LoRA because it allows the adaptation to capture more complex structures (higher-rank approximations can approximate arbitrary weight updates better). (ii) Computational cost per forward pass slightly increases because each forward pass now includes additional matrix multiplications BA involving larger matrices. However, as long as $r \ll d$, the additional computational cost remains relatively small compared to full fine-tuning.*

Your answer:

1.4 (3 points) Suppose during experimentation you find that using very low-rank LoRA updates leads to degraded task performance compared to full fine-tuning. Provide one clear reason for this behavior, and suggest one practical way to mitigate it without significantly increasing parameter counts.

Solution: *Low-rank approximations might overly constrain updates, failing to capture nuanced task-specific details, thus leading to poorer adaptation quality. A practical mitigation is applying LoRA selectively to the most influential layers (e.g., only the query and value matrices of attention layers), allowing these limited parameters to adapt more effectively without increasing the total parameter count drastically.*

Your answer:

1.5 (3 points) Explain clearly why, even though LoRA reduces the memory required for storing adapted parameters, it generally does not significantly decrease the number of floating-point operations (FLOPs) needed for inference.

Solution: *LoRA does not significantly reduce FLOPs during inference because the adapted weights still require matrix multiplications identical in size and complexity to those of the original model. Although the updates themselves have fewer parameters, the combined forward pass ($W + BA$) does not simplify computationally, as the effective dimensions remain unchanged.*

Your answer:

2. Now Prabhav is adapting a pre-trained LM for sentiment classification via prompt tuning (soft prompts). The dataset is small,and he needs to adapt quickly, without retraining or storing multiple large models.

2.1 (3 points) Practically, you notice that prompt tuning significantly underperforms compared to full fine-tuning for smaller language models but performs similarly on larger models. Explain this observation using insights from model capacity and adaptability.

Solution: *Smaller language models have limited capacity and rely more heavily on significant parameter adjustments to adapt to new tasks. Prompt tuning, adjusting only a small set of parameters, provides insufficient flexibility for smaller models. Larger models, however, possess greater representational capacity, enabling them to generalize well from limited prompt tuning, thus closely matching the effectiveness of full fine-tuning.*

Your answer:

2.2 (3 points) Suppose you increase the number of soft-prompt tokens to improve model performance. Explain why performance improvements typically plateau after a certain prompt length, and suggest a principled method to choose an appropriate prompt length.

Solution: *Performance improvements plateau because adding more soft prompts beyond a certain point offers diminishing returns—additional parameters offer minimal incremental gains once the model captures task-specific nuances. Mathematically, each additional token's incremental performance improvement becomes negligible (approaches zero). A principled method for choosing prompt length could involve systematically increasing the length while validating against a held-out dataset, selecting the minimal length at which validation accuracy plateaus.*

Your answer:

2.3 (3 points) (Extra Credit) Explain the key difference between prefix-tuning and prompt-tuning in terms of how they modify the model's input and processing. Discuss one theoretical advantage of prefix-tuning for generative tasks like summarization.

Solution: *Prompt-tuning prepends trainable embeddings to the input sequence, affecting only the initial input to the model. Prefix-tuning, however, adds trainable key-value pairs to the attention mechanism at every transformer layer, influencing the model's contextual processing throughout. A theoretical advantage for generative tasks like summarization is that prefix-tuning can guide attention at multiple layers, enabling finer control over long-range dependencies and coherent output generation, which is critical for producing concise and contextually relevant summaries.*

Your answer:

Q4. Alignment *16 (+4 bonus) points*

1. (4 points) You apply on-policy RLHF directly to a base LM (i.e., collect human feedback on candidate generations of this base LM) that has not undergone instruction tuning, using a reward model perfectly aligning with human preferences in open-ended dialogue. What outcomes might you observe? In this setting, is supervised fine-tuning (SFT) still necessary before RLHF?

Solution: *Without SFT, the model may generate incoherent or off-topic responses, making human preference comparisons noisy or meaningless. SFT provides a baseline level of instruction-following ability, ensuring that RLHF fine-tunes quality rather than fixing basic behavior.*

Your answer:

2. (4 points) In a separate scenario, suppose you train a LM with Policy Gradient but disable the KL-divergence penalty just to see what will happen. You observe that trained LM achieves high reward, producing long, fluent outputs that are nonetheless semantically empty. What happens in this case, and is this primarily caused by the RLHF algorithm, the reward model, the absence of penalty, or some combination of these?

Solution: *This illustrates reward hacking under Goodhart's law: once the reward proxy becomes the target, it no longer tracks true quality. By removing the KL penalty that kept the fine-tuned policy close to the pretrained LM's natural-language distribution, the model drifts into exploiting quirks of the learned reward function—producing fluent but semantically empty text to maximize proxy reward. The reward model might overvaluing superficial qualities like fluency or length bias, rather than meaningful or accurate content. Adding back the KL penalty helps mitigate this issue by constraining the policy.*

Your answer:

3. (4 points) In the DPO loss

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\left(\log\frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \log\frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right)\right)\right],$$

where π_θ is the learned policy and π_{ref} is the referenced policy. What does the inner log-ratio difference represent in the loss? Which part of the original RLHF objective does π_{ref} correspond to? Also briefly explain how the parameter β affects model alignment.

Solution: *The log-ratio difference compares how much more the current policy prefers the better response than the worse one, relative to the reference policy. It reflects the model's relative preference shift. The parameter β controls the sharpness of this comparison: larger leads to stronger preference alignment, while smaller smooths the update. The reference model π_{ref} corresponds to the KL-penalty term in the standard RLHF objective — it acts as the anchor that prevents the fine-tuned policy π_θ from drifting too far from the base LM, preserving language fluency and grounding.*

Your answer:

4. (4 points) After GPT-5 is pretrained, you're tasked with aligning it using RLHF. To reduce annotation and generation costs, you consider reusing old comparison data—(response, reward) pairs collected from users choosing between generations by earlier models like GPT-3.5 for train your reward model. Your teammate insists on gathering new comparisons from GPT-5 instead. What benefit are they aiming for, and what issue might arise if you continue using this data for GPT-5–and eventually GPT-100?

Solution: *They aim to ensure that training data matches the current model's behavior, avoiding stale or misleading signals. Reusing older data introduces distribution drift—rewards reflect past policies, not the current one—leading to off-policy mismatch and degraded learning.*

Your answer:

5. (4 points) (Extra credit) Because collecting large, diverse, human-annotated instruction–response data is prohibitively costly and slow, you want to create an automated pipeline that lets a base language model generate, assess, and refine its own training examples for alignment—can you propose one such pipeline and explain how each of its components contributes to effective, scalable alignment?

Solution:

Your answer:

Q5. Inference-time Scaling *8 points*

1. (4 points) You deploy a 7B-parameter "thinker" model on a reasoning benchmark. Because your training

budget is fixed, you decide to trade inference compute for quality by running Best-of-N rejection sampling with a verifier that returns a binary outcome reward (correct /incorrect). After $N = 32$ the accuracy curve flattens even though you continue to increase N. Explain why accuracy saturates despite more samples, and name two changes that could unlock further gains without retraining the model.

Solution: *Accuracy saturates because candidates sampled from the same policy often share correlated errors, leading to repeated mistakes. To address this, two strategies can be employed: first, introduce controlled diversity by increasing the sampling temperature or using nucleus sampling to generate candidates that explore different reasoning trajectories; second, replace rejection sampling with step-wise search methods like Monte Carlo Tree Search (MCTS) or Beam Search, leveraging a verifier to evaluate partial solutions and guide the search toward more promising reasoning paths.*

Your answer:

2. (4 points) Suppose you're prompting a a decoder-only Transformer LM. Now consider two modifications: (i) increasing the length of the *input* prompt by one token, and (ii) increasing the length of the generated *output* by one token. Which of these changes would result in higher latency (and why)?

Solution: *Generating an extra output token consumes more latency than appending a token to the input prompt. Generating tokens are sequential while inputs are read in parallel.*

Your answer:

Q6. Training Efficiency *32 points*

1. (4 points) During neural network training component consumes the most memory in a GPU during the forward pass and which component consumes the most memory during the backward pass?

Solution: *Forward: Activations; Backward: Optimizer States*

Your answer:

2. (4 points) Match (draw a line) connecting each NCCL operation with its correct description.

Operations	**Descriptions**
1) `all_reduce`	A) Each GPU stores the sum of a shard of the input
2) `reduce_scatter`	B) Every GPU performs a broadcast operation in parallel
3) `all_gather`	C) Sum over input tensors, then duplicate to all GPUs
4) `broadcast`	D) Duplicates one tensor to all GPUs

Solution:

(a) all_reduce - C) Sum over input tensors, then duplicate to all GPUs

(b) reduce_scatter - A) Each GPU stores the sum of a shard of the input

(c) all_gather - B) Every GPU performs a broadcast operation in parallel

(d) broadcast - D) Duplicates one tensor to all GPUs

3. (4 points) Daniel is training his own Language Model called DanielGPT, and he's facing memory constraints. His model has 1 billion parameters and he's training with Adam Optimizer. When using naive data parallelism across 8 GPUs:

- Each parameter requires 2 bytes for FP16 model parameters
- Each parameter requires 2 bytes for FP16 gradients
- Each parameter requires 4 bytes for FP32 Adam first-order momentum
- Each parameter requires 4 bytes for FP32 Adam second-order momentum

Hint: 1GB $= 1 \times 10^9$ bytes

Calculate the memory required *per GPU* with naive data parallelism (ignoring activations).

Solution: $(2 + 2 + 4 + 4) \times 1 * 10^9 = 12 * 10^9 bytes = 12GB$

Your answer:

4. (4 points) Taking the same setup as the previous question, Daniel wants to save some memory by using ZeRO-1 optimizer. Calculate the memory required per GPU if Daniel implements ZeRO-1 (again, ignoring activations).

Solution: $(2 + 2 + \frac{4+4}{8}) * 1 * 10^9 = 5 * 10^9 bytes = 5GB$

Your answer:

5. (3 points) Amazed by the memory savings, Daniel decided to implement ZeRO-1 with 8 GPUs. Unfortunately he forgot the NCCL operations to implement them. (face palm!) Fill in the blanks to help Daniel figure out which NCCL operation he needs to use in each of the following steps (a) *During backpropagation, gradients are split across GPUs using _____ NCCL operation, with each GPU collecting and aggregating only the gradient components corresponding to the optimizer states it's responsible for storing.* (b) *Each GPU updates part of the parameters and uses the _____ NCCL operation to synchronize its updated gradients with every other GPU.*

Solution: *reduce_scatter, all_gather*

6. (4 points) In basic pipeline parallelism (microbatches=1), if you split a model across 4 GPUs (PP=4), what fraction of time is each GPU working and why?

Solution: *Each GPU is working only 1/4 of the time because it needs to wait for other GPUs to finish forward and backward passes.*

Your answer:

7. (3 points) Two common reduced precision formats are FP16 (16-bit floating point) and BF16 (Brain Floating Point 16-bit). FP16 uses 1 sign bit, 5 exponent bits, and 10 mantissa bits, while BF16 uses 1 sign bit, 8 exponent bits, and 7 mantissa bits. Explain the primary advantage of using BF16 over FP16 in training large language models.

Solution: *The primary advantage of BF16 over FP16 in training large language models is its wider range (due to 8 exponent bits) but sacrificing precision. The former allows for better numerical stability during training, especially without the need for loss scaling.*

Your answer:

8. (6 points) Standard Knowledge Distillation minimizes the Cross-Entropy between the student's predicted probability distribution $p_s(\cdot \mid x)$ and the teacher's predicted probability distribution $p_t(\cdot \mid x)$ for a given input x over all labels $y \in \mathcal{Y}$: $\mathcal{L} = -\sum_{y \in \mathcal{Y}} p_t(y \mid x) \log p_s(y \mid x)$. Daniel found a very good model on the web developed by ClosedAI Inc called "ClosedGPT" and he wants to distill from ClosedGPT to his student model DanielGPT. Unfortunately, he does not have access to the probabilities of ClosedGPT, only the textual outputs of ClosedGPT can be seen.

   (a) (3 points) Describe a way that Daniel can still distill from ClosedGPT, and describe how it differs from standard distillation.

   Solution: *Daniel can use sequence-level knowledge distillation instead of standard distillation. In this approach, he would: Generate a large dataset of input prompts For each prompt, collect the output generated by ClosedGPT Train his DanielGPT model to predict these generated outputs given the same prompts.*

   Your answer:

   (b) (3 points) Usually the cost of ClosedGPT is high, but now ClosedAI offers **infinite** access to ClosedGPT. Show that you can approximate the standard Knowledge distillation giving this infinite access.

   Solution: *While Daniel can't directly access $p_t(\cdot \mid x)$, he can approximate it by requesting many samples from ClosedGPT for the same input x The approximation works as follows:*
   - For each training input x, generate N samples from ClosedGPT (where N is very large, possible with infinite access)
   - Create an empirical distribution $\hat{p}_t(\cdot \mid x)$ by counting the frequency of each token y that appears as the next token across all N samples. As N approaches infinity, $\hat{p}_t(\cdot \mid x)$ will converge to the true $p_t(\cdot \mid x)$ by the law of large numbers.

Your answer: