Homework-2
11-664/763: Inference Algorithms for Language Modeling
Fall 2025

Instructors: Graham Neubig, Amanda Bertsch

Teaching Assistants: Clara Na, Vashisth Tiwari, Xinran Zhao

Due: October 28th, 2025

Instructions

Please refer to the collaboration, AI use policy as specified in the course syllabus.

1 Shared Tasks

Throughout the semester, you will be working with data from three shared tasks. We host the data for each shared task on Hugging Face; you can access them at **[this link]**. We will generally ask for results on the "dev-test" split, which consists of 100 examples for each task, using the evaluation scripts provided. The remainder of the examples can be used for validation, tuning hyperparameters, or any other experimentation you would like to perform. The final shared task at the end of the semester will be evaluated on a hidden test set.

Algorithmic The task that the language model will tackle is N-best Path Prediction (Top-P Shortest Paths). Given a directed graph $G = (V, E)$ with $|V| = N$ nodes labeled $0, \ldots, N-1$ and non-negative integer edge weights $w : E \to 1, \ldots, W$, the task is to find the top-P distinct simple paths from source $s = 0$ to target $t = N - 1$ minimizing the additive cost

$$c(\pi) = \sum_{(u,v)\in\pi} w(u, v). \tag{1}$$

The output is a pair

$$\texttt{paths} = [\pi_1, \ldots, \pi_P], \quad \texttt{weights} = [c(\pi_1), \ldots, c(\pi_P)], \tag{2}$$

sorted by non-decreasing cost. The language model will be expected to use tool calls[1] to specify its answer.

Evaluation compares predicted pairs $(\pi, c(\pi))$ against the reference set with the score

$$\text{score} = \frac{|(\pi, c(\pi))\text{pred} \cap (\pi, c(\pi))\text{gold}|}{P}. \tag{3}$$

[1] https://platform.openai.com/docs/guides/function-calling

MMLU medicine We will use the two medicine-themed splits of MMLU: college_medicine and professional_medicine. Evaluation is on exact match with the correct multiple-choice answer (e.g. “A”).

Infobench Infobench provides open-ended queries with detailed evaluation rubrics. Evaluation **requires calling gpt-5-nano**; we expect that the total cost for evaluation for this homework will be substantially less than $5. See the paper for more information.

2 Best-of-n and MBR [30 points]

In this section, you will be asked to implement, experiment with, and reflect on both best-of-n sampling [1, 8] and Minimum Bayes Risk (MBR) decoding [2, 6] methods. These are methods for reranking *multiple* outputs for the same input.

You will be asked to submit relevant code.

2.1 Warm up

For this section, we'll be reranking a set of outputs for InfoBench. We will release a set of 50 outputs generated with vLLM and `Qwen3-4B` with a temperature setting of 0.2 for each of the 100 examples in the `dev_test` split of the Infobench shared task, along with the InfoBench evaluation score (using GPT-5-nano as the judge) for each of the outputs.

What is the mean, median, and standard deviation of the number of unique generations per prompt? What is the average difference in InfoBench score between the best and worst completion for each prompt? Take a look at some of your generations and write just a sentence or two about what you observe (feel free to pick from: How do the generations tend to differ when they do? Do they vary in length? Are there generations that different in exact match but semantically similar? Are there any types of tasks or examples that seem to lead to more or less diverse outputs?)

Deliverables: code (submitted together with other parts of Q2 via `rerank-outputs.py`), three numbers (mean, median, and standard deviation), and your reflection sentence(s).

Solution:
The mean number of unique generations per prompt is approximately 39.91, with a median of 50.0 and a standard deviation of approximately 16.80. The average difference in InfoBench score between the best and worst completion for each prompt is approximately 0.723. Observations of a sample of generations indicate variations in length and structure across different task types. While some generations for the same prompt share similar initial phrasing, semantic differences are present, as few generations are formatted differently. However, the generations generally adhere to the requested format and content. For factual and code generation tasks, there is often a more constrained set of correct or highly similar answers, leading to less semantic variance. Creative tasks, by nature, allow for a wider range of interpretations and expressions, resulting in more diverse outputs in both length and semantic content. It makes sense that you would observe this difference in diversity based on the task type. □

2.2 Implementing methods for choosing the top output

Now we have 50 (not necessarily unique) outputs for each prompt. For this question, you'll be asked to implement a variety of methods for selecting the single best output, given a list of 50 outputs and possibly the input prompt. All of these methods should be implemented in a file, `rerank_outputs.py`, which you will submit along with this homework. Each method should return **the score for every candidate output**, in the same order that the outputs were passed to the method.

1. **Log probability:** The simplest option is to use the log probability under our model. Implement a method, `compute_model_prob(outputs, prompt)`, which computes the log-likelihood of each output.[2]

[2]Note: you may already have the log-probs for each output from your original generation of the outputs. If so, you can validate that this method returns log-probs *close* to those you have saved, though they may not match exactly if you used an

2. **A stronger model's log probability:** Extend your method above to take an optional third parameter, `model`, with the default being `Qwen3-4B`. For the stronger model, use `Qwen3-14B`.[3]

3. **Scalar reward model:** A scalar reward model is a function that maps state-action pairs to real-valued rewards. Scalar reward models are trained to automatically evaluate LLM outputs (typically with conditioning on inputs, and over entire sequences). Write a method, `compute_scalar_reward(outputs, prompt)` that uses `Skywork/Skywork-Reward-Llama-3.1-8B-v0.2` [10] to compute scalar rewards based on both the input and output text.

4. **Pairwise reward:** One other common way to automatically evaluate LLM outputs is to compare them pairwise using a reward model. Use `llm-blender/PairRM` [9] for this question. Take the score for each output to be the number of other outputs it beats in a pairwise comparison.[4]

5. **MBR with BLEU:** Now, we'll consider computing Minimum Bayes Risk (MBR) instead of using a reward model. Write a method, `mbr_bleu`, that uses MBR with the metric BLEU to rank outputs. You may use an existing implementation of BLEU.

6. **MBR with BertScore:** Write a method, `mbr_bertscore`, that performs MBR as above but using the neural metric BERTScore [17]. You may use an existing implementation of BERTScore (e.g. the one in huggingface).

Deliverables: a file, `rerank_outputs.py`, implementing the above methods.

2.3 Improving efficiency for BERTScore-based MBR

By looking at the way BERTscore is computed, we can devise two tricks to make BERTScore-based MBR more efficient. You do *not* have to implement either of these improvements (although your BERTScore MBR implementation will run faster if you do!).

2.3.1 Reducing the number of comparisons

First, write out the BERTscore equation. For a set of n documents, how many comparisons (i.e. computing BERTScore(A, B)) would you need to make to run MBR, in the naive implementation? How could you reduce the number of comparisons? Give both answers in terms of n, and explain how the reduction in number of comparisons is possible.

<u>**Solution:**</u>

BERTScore Equation:

$$P(H,R) = \frac{1}{|H|}\sum_i \max_j \cos(x_i, y_j), \quad R(H,R) = \frac{1}{|R|}\sum_j \max_i \cos(x_i, y_j), \quad F_1(H,R) = \frac{2PR}{P+R}$$

Number of Comparisons in Naive MBR: For n candidate documents $\{d_1, d_2, \ldots, d_n\}$, MBR computes

$$\text{MBR}(d_i) = \frac{1}{n-1}\sum_{j \neq i} \text{BERTScore}(d_i, d_j)$$

efficient inference library for the original generations.

[3] Note that you do not need to specify thinking or non-thinking mode, as you are not generating any text.

[4] Note that this is only one of several ways of using a pairwise preference model.

A naive implementation requires $n(n-1)$ comparisons (i.e., computing BERTScore(A, B) for all ordered pairs). Since BERTScore(A, B) = BERTScore(B, A), the number of unique comparisons can be reduced to $\frac{n(n-1)}{2}$. □

2.3.2 Reducing the number of forward passes

A naive implementation of MBR with BERTScore requires $O(N^2)$ BERT forward passes. The second trick reduces the number of forward passes to only $O(N)$ BERT forward passes. Explain how this is possible.

Solution:
Each BERTScore(d_i, d_j) run encodes both d_i and d_j, leading to $O(n^2)$ BERT forward passes. However, by precomputing embeddings once for each candidate:

$$E_i = \text{BERT}(d_i)$$

and reusing them to compute pairwise cosine similarities, the number of expensive BERT forward passes reduces to $O(n)$, while maintaining $O(n^2)$ lightweight similarity computations. □

Deliverables: The written responses above.

2.4 Comparing methods for reranking

Now, run your methods above on your precomputed set of InfoBench outputs. **Save the scores** for each output according to each method; you may find them useful for the next problem. Complete the table below. "Oracle" refers to using the gold scores, and is the skyline for performance. Using `Qwen3-4B` log-probs is the baseline. If there is a tie in top score for a method, compute the top-1 score by randomly selecting one of the top-scoring outputs.

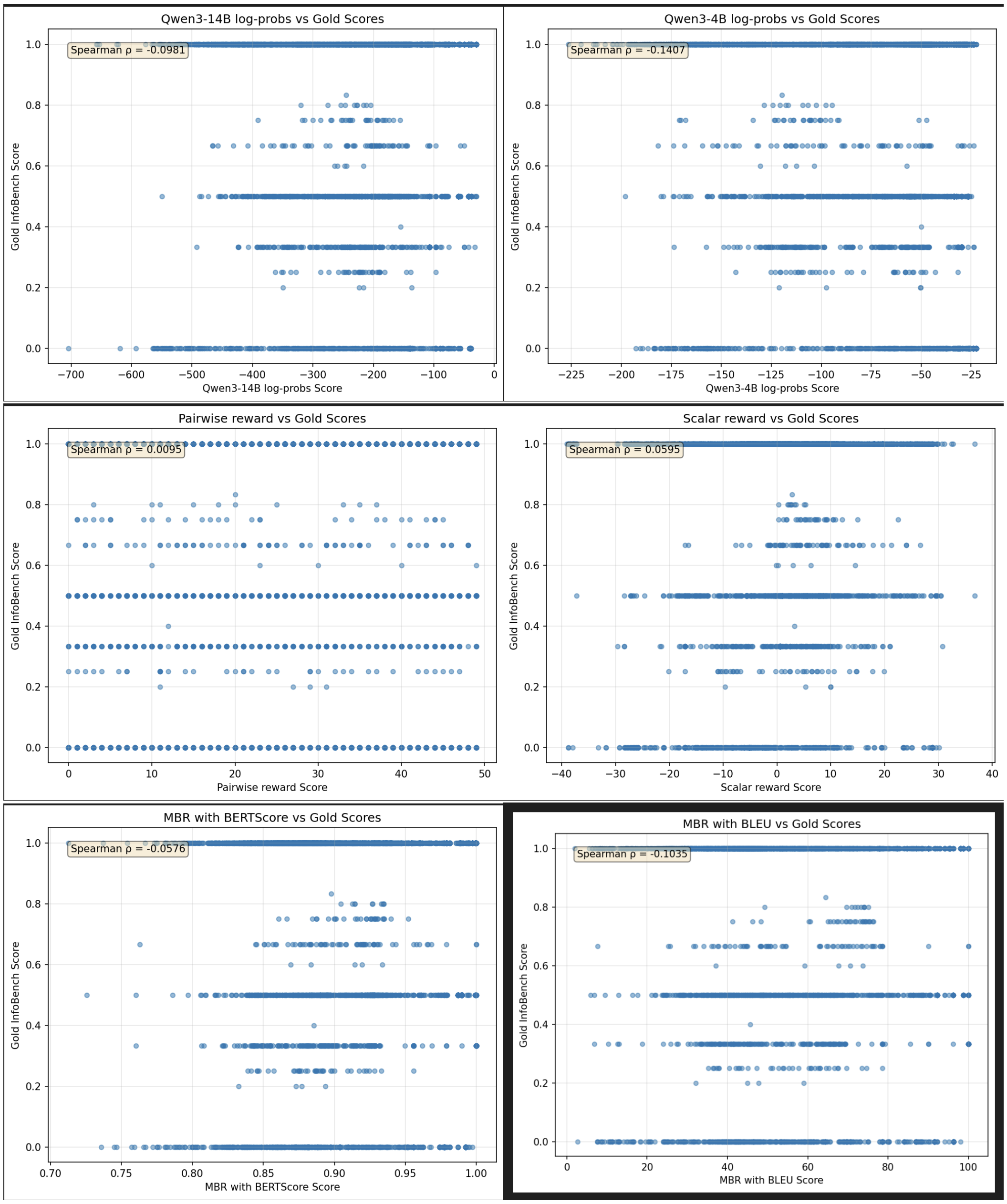
Solution:

Method	Top-1 score	Avg. rank of best output	Spearman rank correlation
Oracle	0.9833	1	1.0
`Qwen3-4B` log-probs	0.7238	29.39	-0.1407
`Qwen3-14B` log-probs	0.7272	28.49	-0.0981
Scalar reward	0.7458	28.73	0.0595
Pairwise reward	0.7893	27.41	0.0095
MBR with BLEU	0.7642	26.19	-0.1035
MBR with BERTScore	0.7417	26.87	-0.0576

□

Additionally, for each method (other than oracle), plot the scores according to that method versus the gold scores in a scatter-plot below. You should have six scatter-plots, one per method.

Solution:



Discuss your findings in terms of average-case and worst-case performance. Which method would you choose for this task based on performance? Justify your answer.

Solution:

Looking at the results, pairwise reward achieves the highest Top-1 score of 0.7893, meaning it successfully picks the best output approximately 79% of the time, outperforming all other methods. Scalar reward follows with 0.7458, while MBR with BLEU achieves 0.7642. The log-probability methods show relatively lower Top-1 scores, with Qwen3-4B at 0.7238 and Qwen3-14B at 0.7272, but interestingly have the worst average ranks (29.39 and 28.49 respectively), suggesting they struggle with overall ranking consistency despite reasonable top-1 performance. MBR with BERTScore performs moderately at 0.7417 but achieves a better average rank of 26.87. The MBR methods generally achieve better average ranks (around 26-27) compared to log-probability and reward methods. However, all methods show poor Spearman correlations with the gold scores (ranging from -0.14 to 0.06), indicating this is a challenging task overall where no method reliably ranks outputs in the same order as human judgment. Based purely on performance, pairwise reward would be the method of choice, as it most frequently identifies the best output and maintains a reasonable average rank of 27.41. □

Discuss your findings in terms of the resources required for each task. You can speak qualitatively (e.g. discuss your impression of relative compute or wall-clock time required, without providing exact measurements). Which method would you choose for this task if you had to balance performance and efficiency? Justify your answer.

Solution:

From a computational efficiency perspective, the log-probability methods are significantly more resource-efficient than the alternatives, requiring only O(n) forward passes through the model. Qwen3-4B is particularly lightweight given its smaller 4B parameter size, making it fast to run even on modest hardware. In contrast, MBR-based methods are far more expensive, requiring O(n^2) pairwise comparisons between all outputs. While MBR with BLEU is relatively cheap per comparison (just string matching), MBR with BERTScore is especially costly, needing to compute neural embeddings even with optimization tricks. The pairwise reward model also suffers from O(n^2) complexity, requiring expensive model evaluations for every pair of candidates, making it impractical for large candidate sets despite its superior Top-1 score of 0.7893. Scalar reward models fall in between, requiring O(n) passes through an 8B model, offering better performance (0.7458) than log-probability methods with moderate computational overhead. When balancing performance and efficiency, scalar reward emerges as the optimal choice. It achieves the second-best Top-1 score after pairwise reward (0.7458 vs 0.7893), while maintaining linear O(n) complexity that scales gracefully with candidate set size. This represents a strong middle ground between the faster but lower-performing log-probability methods and the more accurate but computationally prohibitive pairwise reward approach. □

Compute the length for the top-1 output for each example according to each method. Do you notice length biases in any of the methods?

Solution:

Method	Length of top-1 output
Oracle	196.2
`Qwen3-4B` log-probs	173.04
`Qwen3-14B` log-probs	169.82
Scalar reward	189.69
Pairwise reward	194.34
MBR with BLEU	194.68
MBR with BERTScore	192.89

The length analysis reveals clear biases across different methods. The log probability methods (Qwen3-4B at 173, Qwen3-14B at 169) strongly prefer shorter outputs compared to the Oracle (196).

This occurs because log probs are summed across tokens, since individual token log-probabilities are negative, longer sequences accumulate more negative values, resulting in lower overall scores and an inherent brevity bias. In contrast, MBR methods favor longer outputs (BLEU at 195, BERTScore at 193) because longer responses provide more content for similarity matching. BLEU especially benefits from this through increased n-gram overlap opportunities. The reward models (scalar at 190, pairwise at 194) also trend toward longer outputs, approaching the Oracle length. Given that the Oracle's average length is 196, the log-prob methods may be systematically favoring overly concise responses that lack the comprehensiveness of truly high-quality outputs. The MBR and reward-based methods better approximate the ideal output length. □

Deliverables: the graphs and analysis in this section. You do not need to provide your graphing code for this problem with your homework solutions.

2.5 Varying n

Starting from your original set of 50 output generations, subsample sets of 5, 10, and 20 outputs per model by random selection.

For some (but not all) of the reranking methods above, you can reuse the scores you computed above for the outputs in your smaller set. Which methods do you need to recompute scores for?

Solution:
Pairwise reward (PairRM) - needs pairwise comparisons within the subset
MBR with BLEU - compares each output to all other candidates
MBR with BERTScore - compares each output to all other candidates □

Now, fill out the tables below for your smaller subsets, recomputing reranking methods where necessary.

Solution:
n=5:

Method	Top-1 score	Avg. rank of best output	Spearman rank correlation
Oracle	0.9433	1	1.0
`Qwen3-4B` log-probs	0.7642	2.84	0.095
`Qwen3-14B` log-probs	0.766	2.85	0.0859
Scalar reward	0.7916	2.89	0.0859
Pairwise reward	0.8016	2.94	0.042
MBR with BLEU	0.7916	3.05	0.044
MBR with BERTScore	0.776	2.95	0.1089

n=10:

Method	Top-1 score	Avg. rank of best output	Spearman rank correlation
Oracle	0.9083	1	1.0
`Qwen3-4B` log-probs	0.7435	4.81	0.0982
`Qwen3-14B` log-probs	0.763	4.62	0.102
Scalar reward	0.793	5.25	0.116
Pairwise reward	0.779	5.76	-0.043
MBR with BLEU	0.746	5.47	0.0553
MBR with BERTScore	0.766	5.4	0.091

n=20:

Method	Top-1 score	Avg. rank of best output	Spearman rank correlation
Oracle	0.9767	1	1.0
`Qwen3-4B` log-probs	0.748	9.0	0.084
`Qwen3-14B` log-probs	0.755	8.85	0.095
Scalar reward	0.736	10.5	0.068
Pairwise reward	0.7825	11.22	-0.031
MBR with BLEU	0.7425	9.95	0.055
MBR with BERTScore	0.749	9.77	0.065

□

Discuss. How do trends vary with the size of the set n? How much does scaling up n increase the oracle score? If you had to set an n for performing best-of-n or MBR on this model and dataset, what would you choose, and why?

Solution:

The oracle score shows interesting patterns as n increases. Starting at 0.9433 for n=5, it drops to 0.9083 at n=10 (likely due to random sampling), then jumps significantly to 0.9767 at n=20, and finally increases only slightly to 0.9833 at n=50. This demonstrates diminishing returns, where most of the quality improvement happens between n=5 and n=20, with minimal gains beyond that. As n increases, the average rank of the best output also increases across all methods, meaning it becomes harder to identify the truly best candidate from larger pools. The Spearman correlations remain poor for all methods regardless of n, showing that accurate ranking is challenging at any pool size. For choosing an optimal n value, the computational complexity of each method matters significantly. Methods like best-of-n using log-probabilities have O(n) complexity, so using n=50 is reasonable since the cost scales linearly. However, pairwise reward models and MBR methods (both BLEU and BERTScore) have O(n^2) complexity, making n=50 prohibitively expensive. Since the oracle score improves only marginally from n=20 to n=50, the recommended choice for quadratic-complexity methods is n=10 to 20, which balances good performance with manageable computational cost. For linear-complexity methods, n=50 can be used to capture the full quality potential. Finally, I think the most optimal value of n for all the methods should be between 20 to 50. □

Deliverables: The results and discussion above. You do not need to upload any additional code for this subsection.

3 Self-Refine [30 points]

Recent work in self-refinement has shown that LLMs can iteratively improve their own outputs [11]. *Self-Refine* introduces a closed-loop framework where a model alternates between **generation**, **critique**, and **refinement**, in contrast to human-in-the-loop feedback systems.

In this problem, you will implement self-refinement and investigate factors that affect its efficacy.

NOTE: The repo includes a bare-bones scaffolds. It exists to help you start quickly. Please feel free to change your structure. Any clean, reproducible solution is acceptable.

3.1 Implementation

Please refer to the paper for details on the algorithm. Implement self-refine for the following specifications:

- **Datasets:** `GraphDev` and `MMLU_Med` (`dev_test` splits)
- **Models:** `Qwen/Qwen3-4B` and `Qwen/Qwen3-0.6B`
- **Iterations:** Run up to 4 total steps per example: $i = 1$ (draft) + 3 refinements. Report metrics at each i
- Set random seed 42.

You can utilise remaining splits for validation and to explore different drafting, critiquing, and refinement strategies.

For consistency, run up to 4 refinement iterations.

3.2 Analysis

1. Compare two configurations that may use *different* temperatures for the draft, critique, and refine stages (meaning 2 different runs not 2^3 runs)!

   There is no right answer here, this is for you to see what are the qualities in generation we would want at different stages of self-refinement. Refer to the paper for discussion of this.

   Report the performance of your specifications on the validation set (you can choose a reasonable subset from dev for this task).

   Note: Going forward for the rest of the analysis, you can report the values for the specific temperature settings you choose, no need to sweep over temperatures.

   <u>Solution:</u>
   For Graph Task:

Model	Temps	Draft	Step 1	Step 2	Step 3	Final
Qwen3-0.6B	$(0.7, 1.0, 1.5)$	0.013	0.020	0.007	0.007	0.007
Qwen3-4B	$(0.7, 1.0, 1.5)$	0.073	0.127	0.120	0.140	0.140
Qwen3-0.6B	$(0.1, 1.0, 0.7)$	0.0	0.0	0.0	0.007	0.007
Qwen3-4B	$(0.1, 1.0, 0.7)$	0.093	0.140	0.140	0.153	0.153

   For MMLU Task:

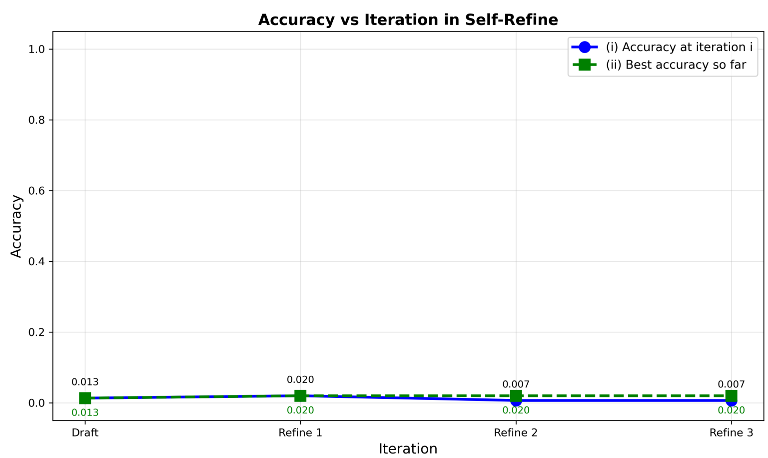
Model	Temps	Draft	Step 1	Step 2	Step 3	Final
Qwen3-0.6B	$(0.1, 1.0, 0.7)$	0.220	0.253	0.260	0.287	0.287
Qwen3-0.6B	$(0.7, 1.0, 1.5)$	0.233	0.260	0.280	0.307	0.307
Qwen3-4B	$(0.1, 1.0, 0.7)$	0.207	0.420	0.433	0.460	0.460
Qwen3-4B	$(0.7, 1.0, 1.5)$	0.247	0.593	0.560	0.540	0.540

* I assumed the question asked 2 different runs for each model on each task. Hence, I have carried out all the above experiments. Here, the temperatures are arranged in (draft, feedback, refine) format. □

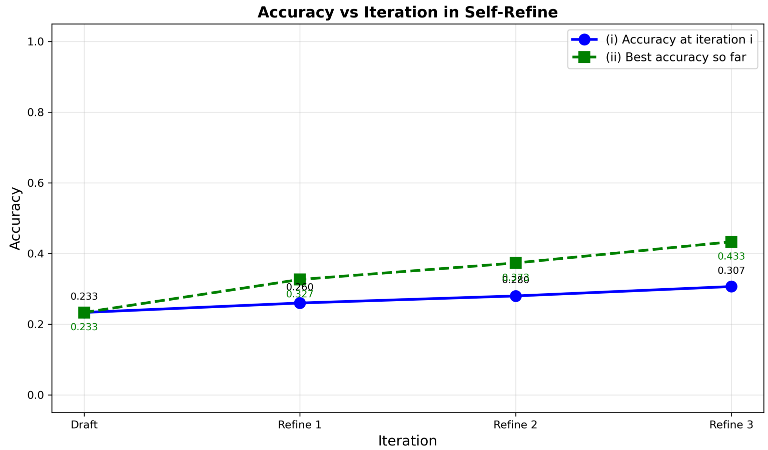
2. Plot how accuracy changes across iterations. Show both (i) accuracy at each iteration i and (ii) the best accuracy so far (i.e., mark an individual example correct if it has been correct for *at least one* iteration).

Solution:

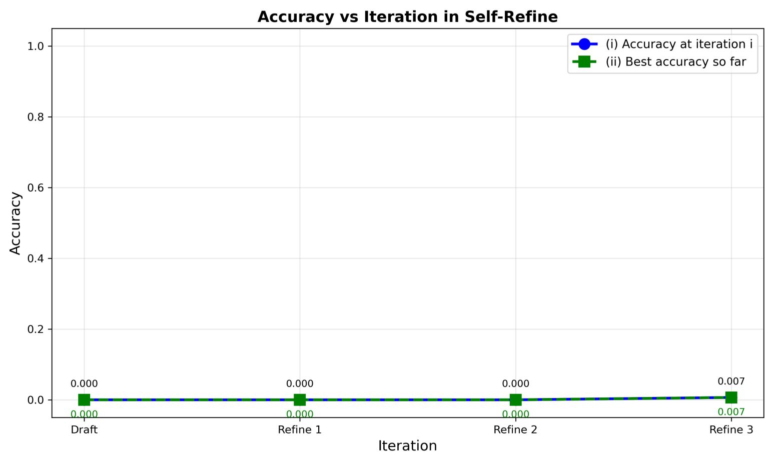
The following are the graphs obtained for different configurations and models: **For Qwen3-0.6B model** with draft temperature: 0.7, feedback temperature: 1.0 and refine temperature: 1.5 => Graph Task —-



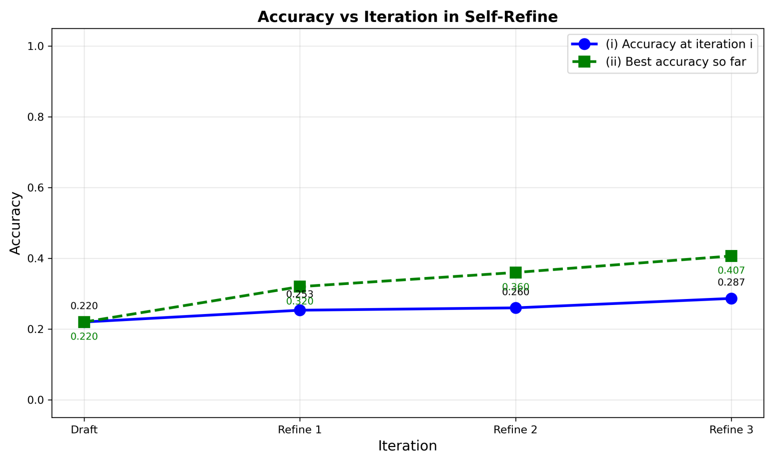
MMLU Task —-



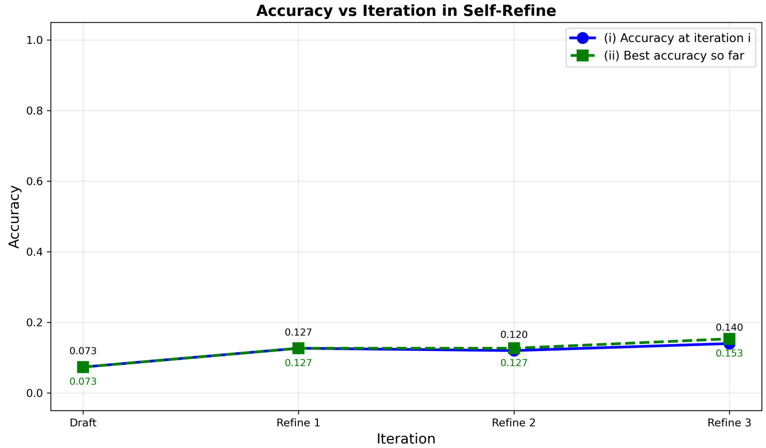
For Qwen3-0.6B model with draft temperature: 0.1, feedback temperature: 1.0 and refine temperature: 0.7 => Graph Task —-



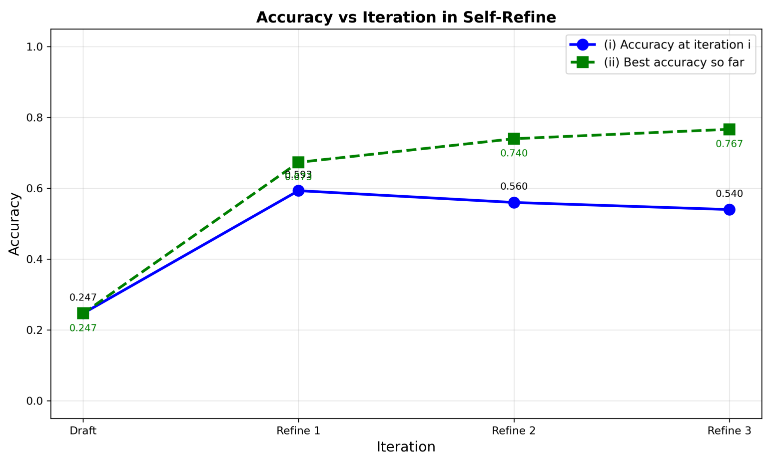
MMLU Task —-



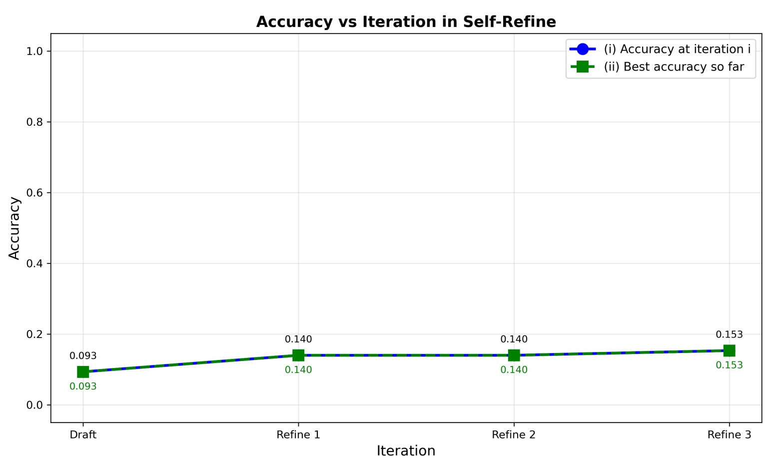
For Qwen3-4B model with draft temperature: 0.7, feedback temperature: 1.0 and refine temperature: 1.5 => Graph Task —-



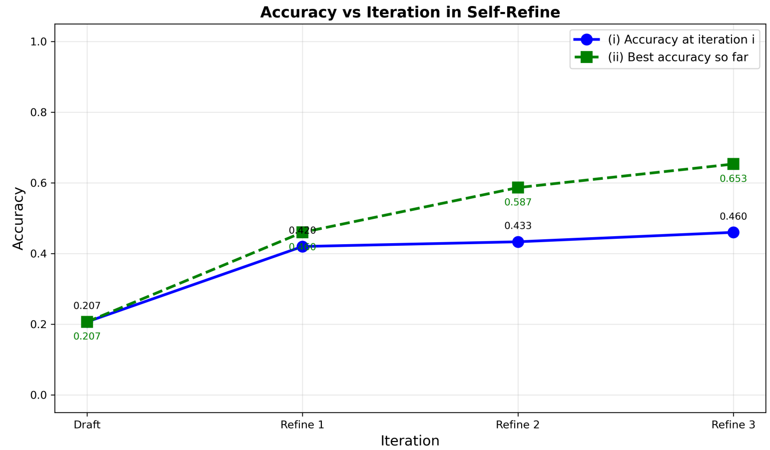
MMLU Task —-



For Qwen3-4B model with draft temperature: 0.1, feedback temperature: 1.0 and refine temperature: 0.7 => Graph Task —-



MMLU Task —-

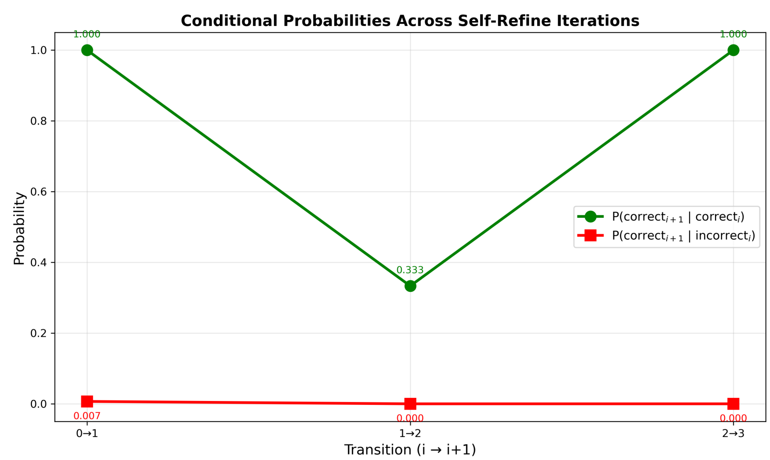


☐

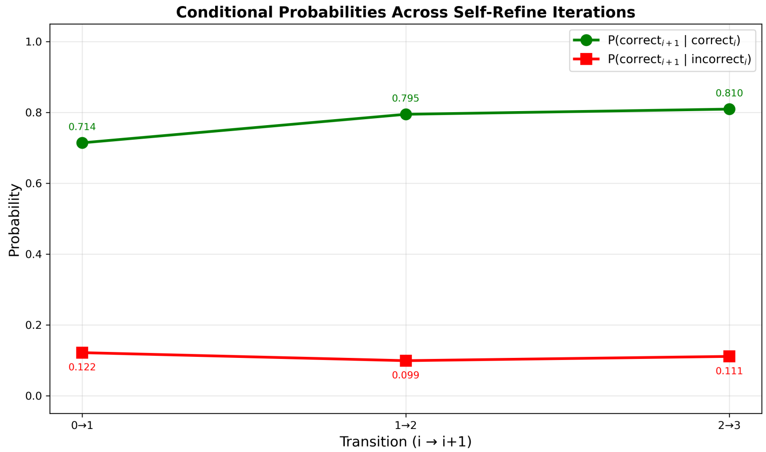
3. Analyze $P(\text{correct}_{i+1} \mid \text{correct}_i)$ and $P(\text{correct}_{i+1} \mid \text{incorrect}_i)$. We want to see how self-refine improves upon an incorrect answer and how it affects where the response was already correct.

   Plot these results and provide at least one example where refinement improves an incorrect answer and one where it harms a correct one (if you see this) for each dataset.

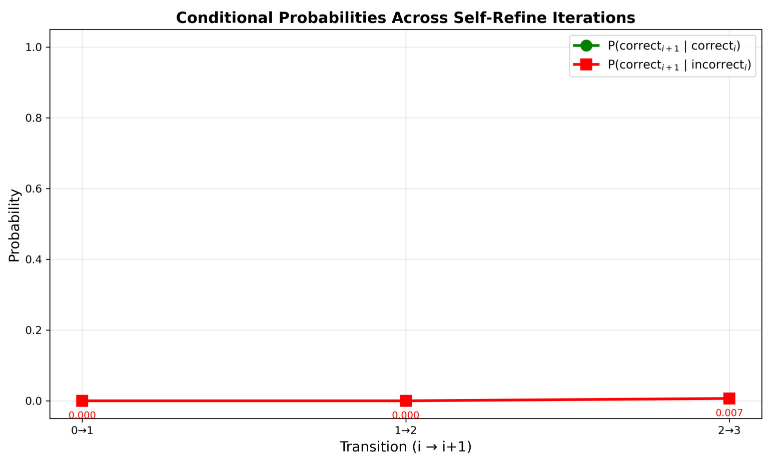
Solution:
The following are the graphs obtained for different configurations and models: **For Qwen3-0.6B model** with draft temperature: 0.7, feedback temperature: 1.0 and refine temperature: 1.5 => Graph Task —-



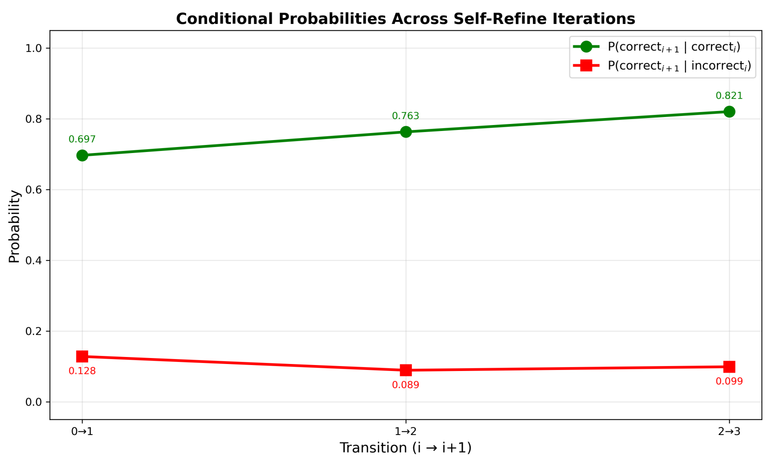
MMLU Task —-



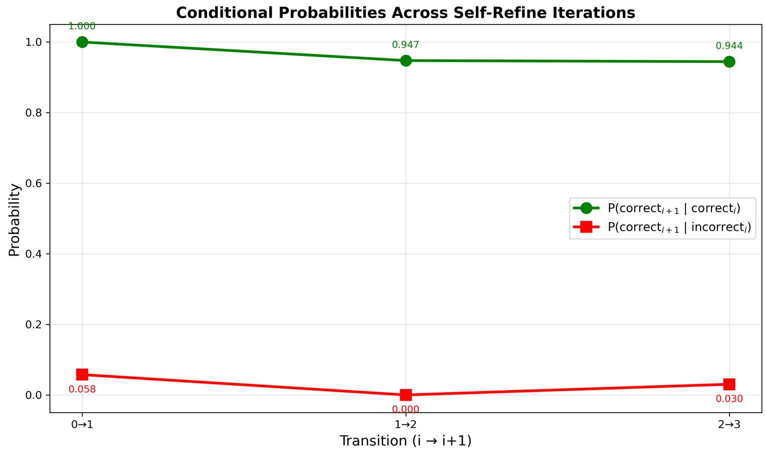
For Qwen3-0.6B model with draft temperature: 0.1, feedback temperature: 1.0 and refine temperature: 0.7 => Graph Task —-



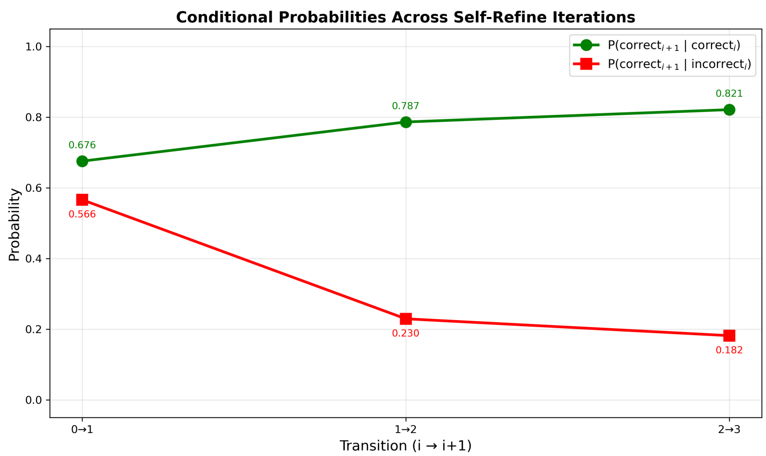
MMLU Task —-



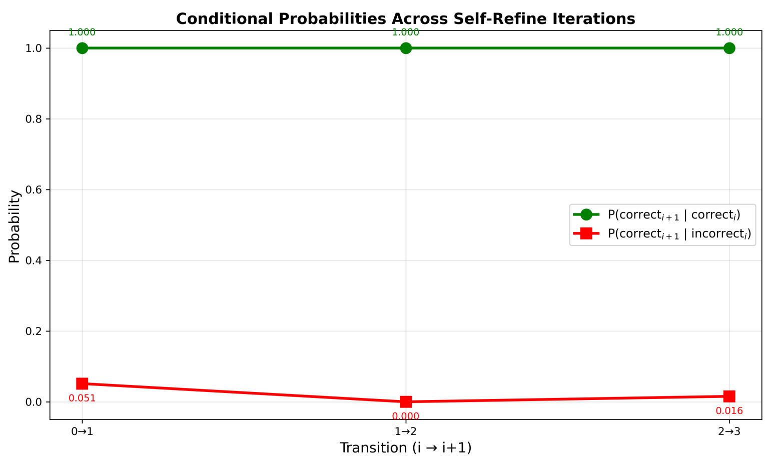
For Qwen3-4B model with draft temperature: 0.7, feedback temperature: 1.0 and refine temperature: 1.5 => Graph Task —-



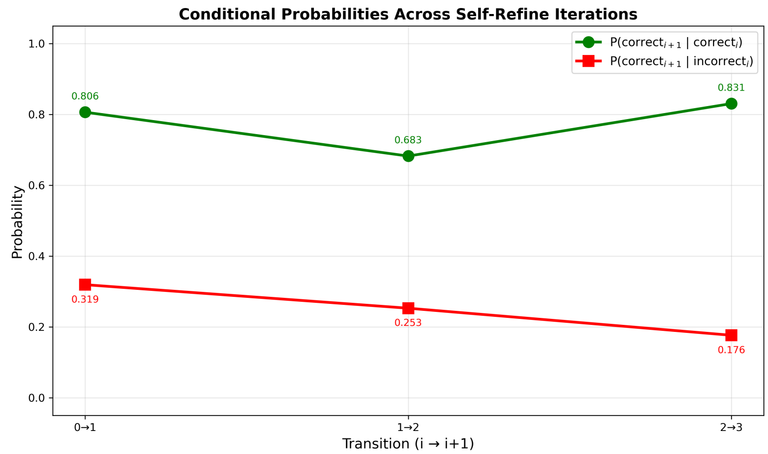
MMLU Task —-



For Qwen3-4B model with draft temperature: 0.1, feedback temperature: 1.0 and refine temperature: 0.7 => Graph Task —-



MMLU Task —-



Refinement Examples Analysis

Graph Task

Example: Refinement Improves an Incorrect Answer

- **Example ID:** 231
- **Question:** Find the single shortest path from node 0 to node 16 in a directed graph with 17 nodes (numbered 0 to 16). The graph contains weighted edges connecting the nodes.
- **Ground Truth:** `{"paths": [{"path": [0, 2, 8, 16], "weight": 123}]}`
- **Draft Answer (Incorrect):**
  - Parsed: `{"paths": [[0, 2, 8, 16]], "weights": [201]}`
  - Path: $0 \to 2 \to 8 \to 16$
  - Calculated weight: 201 (incorrect calculation)
  - Status: Incorrect
- **Feedback Received:**

  "The JSON is valid with proper bracket closure. The path [0,2,8,16] exists in the graph as per the given edges: $0 \to 2$ (weight: 20), $2 \to 8$ (weight: 2), $8 \to 16$ (weight: 101). Total weight = 20 + 2 + 101 = 123, not 201. There is an error in the weight calculation. The correct total weight should be 123, not 201. The path is valid, but the weight is incorrectly calculated."
- **Refined Answer (Step 3):**
  - Parsed: `{"paths": [[0, 2, 8, 16]], "weights": [123]}`
  - Path: $0 \to 2 \to 8 \to 16$
  - Corrected weight: 123
  - Status: Correct
- **Analysis:** The model correctly identified the shortest path but made an arithmetic error when calculating the total weight (201 instead of 123). The feedback explicitly recalculated the edge weights (20 + 2 + 101 = 123) and pointed out the discrepancy. This allowed

the model to correct the weight calculation in the refinement step while maintaining the correct path. This demonstrates how self-refinement effectively catches and corrects computational errors.

MMLU Medical Task

Example: Refinement Improves an Incorrect Answer

- **Example ID:** 144
- **Question:** A high school wrestler is trying to lose 30 pounds over 2 months. Which of the following is NOT a good method to restrict his caloric intake?
  (A) Drink only water.
  (B) Reward himself with a savory meal every Saturday for meeting his calorie goals.
  (C) Have a few pieces of candy when stressed.
  (D) Hide snack food out of sight within his house.
- **Ground Truth:** D
- **Draft Answer (Incorrect):**
  - Answer: A (Drink only water)
  - Status: Incorrect
- **Feedback Received:**

  "After reviewing the options, the most 'not a good method' to restrict a wrestler's caloric intake, given his goal of losing weight (30 lbs over 2 months), is: D. Hide snack food out of sight within his house. Option D is not a good method because it doesn't restrict caloric intake. It focuses on obfuscation of the snack, not restriction. The other options are effective in promoting healthier eating habits or keeping people engaged with their goals."
- **Refined Answer (Step 3):**
  - Answer: D (Hide snack food out of sight within his house)
  - Status: Correct
- **Analysis:** The draft incorrectly identified drinking water as a poor method for restricting caloric intake. The feedback correctly explained that hiding snacks (option D) merely obscures food rather than restricting intake, making it the least effective method. The refinement successfully corrected the answer by recognizing that actual caloric restriction requires more than visual concealment of food.

Example: Refinement Harms a Correct Answer

- **Example ID:** 49
- **Question:** A 32-year-old man presents with difficulty focusing on tasks, increased energy requiring only 4 hours of sleep nightly, rapid speech, and labile affect. He has no history of psychiatric illness and has been substance-free for 3 years. Physical examination and

most cognitive testing are normal. Which of the following is the most likely diagnosis?

(A) Antisocial personality disorder

(B) Attention-deficit/hyperactivity disorder

(C) Bipolar disorder

(D) Borderline personality disorder

- **Ground Truth:** C (Bipolar disorder)
- **Draft Answer (Correct):**
  - Answer: C (Bipolar disorder)
  - Reasoning: Correctly identified decreased need for sleep, increased energy, rapid speech, and labile affect as signs of manic episode consistent with bipolar disorder
  - Status: Correct
- **Feedback Received:**

  "Your answer is correct in identifying C. Bipolar disorder as the most likely diagnosis. However, there are a few improvements and considerations that can be made to make the answer more accurate and complete. [...] Since bipolar disorder is a psychiatric condition that can manifest with mood changes, sleep disturbances, and attentional issues, C is correct. A. Antisocial personality disorder would involve impulsivity, lack of empathy, and behavioral patterns, which do not match the clinical description."
- **Refined Answer (Step 1):**
  - Answer: A (Antisocial personality disorder)
  - Status: Incorrect
- **Analysis:** The draft correctly diagnosed bipolar disorder based on classic manic symptoms (decreased sleep need, increased energy, rapid speech, labile affect). However, the feedback, despite confirming the answer was correct, introduced discussion of alternative diagnoses. This appears to have caused confusion during refinement, leading the model to incorrectly change its answer to antisocial personality disorder. This demonstrates a failure mode where overly elaborate feedback or discussion of ruled-out alternatives can cause the model to second-guess a correct answer, even when the feedback explicitly validates the original choice.

□

4. What differences do you see between the two models for each task? Comment on the initial accuracy, relative improvements (or the lack there of), and stability / response quality over the iterations.

Solution:

For the Graph task, Qwen3-0.6B struggled to generate accurate responses despite the refinement loop, with initial accuracy near zero. Increasing the refinement temperature caused performance degradation rather than improvement. The model showed only minimal gains even after multiple

refinement iterations. In contrast, Qwen3-4B demonstrated substantially better performance, starting with higher initial accuracy and showing consistent improvement through refinement iterations, reaching final accuracies of 0.140-0.153. The larger model maintained more stable accuracy and showed clear signs of benefiting from the feedback loop.
For the MMLU task, both models performed reasonably well, with Qwen3-0.6B showing steady improvement from initial accuracies around 0.220-0.233 to final accuracies of 0.287-0.307. However, Qwen3-4B exhibited an interesting pattern where it achieved high initial performance (jumping to 0.593 after the first refinement) but then degraded in later iterations (dropping to 0.540 by the final step). This degradation appears to stem from verbose feedback that introduced alternative answer choices, causing the model to second-guess its initially correct responses. When the model produced correct answers initially, the feedback loop reinforced those answers, suggesting the model effectively recognizes and promotes patterns from its training data. However, when extensive feedback discussed multiple alternatives, the larger model tended to overthink and change correct answers to incorrect ones, demonstrating a critical failure mode where more sophisticated reasoning capabilities can paradoxically lead to worse performance. □

5. Lastly, what do you think are the shortcomings of this method (in terms of performance, computational cost, etc.)? How can you improve it?

Solution:
The self-refinement method has several significant shortcomings. First, computational cost increases substantially, as each example requires multiple forward passes through the model—one for the initial draft, one for feedback generation, and one for each refinement iteration. This results in approximately 4× the inference cost compared to standard generation, along with increased latency from sequential processing. Second, the method faces context window limitations, as accumulated text from draft, feedback, and refinement stages consumes progressively more tokens. For complex tasks requiring detailed explanations, the growing context can exceed model capacity, forcing truncation of important information. Third, the method exhibits a pattern-matching vulnerability, particularly in smaller models. These models tend to replicate patterns from few-shot examples provided in the draft, feedback, or refinement prompts, leading to performance degradation rather than improvement. This dependency on prompt structure severely limits prompt engineering flexibility, as few-shot learning approaches that typically enhance performance can instead cause the model to produce derivative outputs rather than genuine refinements. Fourth, the method lacks external verification mechanisms, relying entirely on the model to evaluate its own outputs. This self-assessment is particularly unreliable for smaller models that may not have sufficient capacity to accurately critique their own reasoning.
To improve this method, several approaches could be adopted. Using a separate, larger model specifically for feedback generation could provide more reliable critique while keeping refinement costs manageable. Implementing early stopping mechanisms would prevent unnecessary iterations when accuracy plateaus or begins to degrade. Incorporating external verification tools, such as code execution for computational tasks or knowledge base lookups for factual questions, would ground feedback in objective evaluation rather than model self-assessment. Additionally,

applying selective refinement only to low-confidence outputs, rather than refining all responses uniformly, could reduce computational overhead while focusing resources where improvement is most likely. Finally, integrating RAG during the feedback stage could provide factual grounding and reduce hallucination in critique. □

Deliverables

- □ Please include your main self-refine file as `self_refine.py` with exact commands on the different models and temperatures in `run_self_refine.sh/slurm`
- □ Plot(s): Accuracy, best-accuracy vs iter, plots showing $P(\text{correct}_{i+1} \mid \text{correct}_i)$ and $P(\text{correct}_{i+1} \mid \text{incorrect}_i), \forall i \in [4]$
- □ requirements.txt + README
- □ results (output jsons and figures)
- □ Written responses

4 MCP and agentic APIs [30 points]

Recent advances in LLMs have fueled growing interest in building deep research systems that analyze information from various sources and produce a detailed report to answer challenging questions [4, 12, 15, 14]. In this question, we will explore how to use MCP and agentic APIs to build a simple deep research agent. The source code is provided in the `basic_deepresearch` folder. Please follow the `README.md` for detailed instructions and `simple_react.ipynb` for the workflow.

4.1 Implementation: A Basic Deep Research Agent

Your **Task 1** is to fill in some missing code in `react_agent.py`. After that, you can make your agent work! You can play with the pipeline with code under **A basic ReAct Agent** in `simple_react.ipynb`. You can also compare different browse tools by updating `react_agent.yaml` to understand how they work.

4.2 Implementation: Deep Research for MMLU

Upon building up the agent pipeline, we further explore how to leverage the pipeline to help with your shared task. After understanding the pre-implemented search tool, your Task 2.1 is to build an MCP-style tool to support a callable evaluation for the graph shortest path problems, which can help your future Self-Refine pipeline built in a ReAct style.

Next, we move to utilize the deep research agent to answer knowledge-intensive questions. Finish your **Task 2.2**, and then you will be able to extend the MMLU inference pipeline we built previously. Save your best `results_{model}_30_react_agent_mmlu.json` to show us your progress!

Your **Task 3** is to complete some analysis code and report the statistics as follows.

Describe three settings you tried through altering the `react_agent_mmlu.yaml`, e.g., change the base models, number of documents, browse tools, and think-search cycles. Briefly introduce your variants and report the token usage for different steps in the following table (default configuration counts as 1).

Solution:

Describe your variants:

Variant 1: default config (using gpt-4o-mini)

Variant 2: Using crawl4ai as browse tool, increase the browse max pages to 3 and increase the number of thinking search cycles to 4

Variant 3: Using crawlai as browse tool, with max browse pages as 2. Also using serper as the search tool and keeping 4 as the number of documents to search. Finally, increased the number of think search cycles to 3. Also used gpt-4.1-mini as the base model.

A Table for the accuracy and the number of tokens for each variant:

Variant	Acc.	Thinking	Queries	Snippet Titles	Snippet Contents	Final Answers
1	0.83	658.77	24.3	134.4	1076.5	324.47
2	0.567	1504.3	24.77	137.37	1106.93	223.8
3	0.77	986.53	25.97	137.67	1154.33	311.63

□

4.3 Discussion: Evaluation of Long-form Tasks.

Besides the previously discussed short-form answers that are easily verifiable, i.e., with a number or a phrase as the answer. Another important usage of deep research agents is to generate long-form reports for open-ended questions, with statements grounded by the search. For example, generating a multi-section report with citations on the query: *What is deep research?*

Recently, researchers have been working on benchmarking these long-form generation tasks, and have proposed various benchmarks, including but not limited to **AstaBench-SQA-CS-V2** [3] (page 44), **DeepResearch Bench** [5], **DeepScholar Bench** [13], and **ResearchQA** [16]. Read DeepResearch Bench and DeepScholar Bench and answer the following questions:

Q 4.3.1. How is citation evaluated? Write 3-4 sentences for one dataset you chose about the motivation and the design. Visualizations in ALCE [7] can be helpful to build an intuition.

Q 4.3.2. What can be a potential problem of the current evaluation framework with citation or the report generation in general? Write 3-4 sentences for your answer. You should include one problem, one example, and one suggested fix you can think of.

Solution:

Q 4.3.1. Dataset Choice :{DeepScholar Bench} DeepScholar Bench evaluates citation, measuring Citation Precision (whether each citation supports the given claim) and Citation Coverage (whether each claim is fully supported by the cited sources). The motivation is to assess whether generated research synthesis reports are verifiable and properly grounded in sources, which is critical for academic writing tasks like generating related work sections. The design uses LLM-based judges to verify entailment relations between claims and citations, and the human agreement study validates that these automated metrics demonstrate strong agreement with expert human annotators.

Q 4.3.2. Current evaluation framework do not capture the relationships between concepts presented by different citations. For example, when a report synthesizes information by comparing findings across multiple sources ("Method A achieves 90% accuracy [1] while Method B achieves 85% [2], suggesting A outperforms B"), current metrics only verify that each individual citation supports its associated fact, but fail to evaluate whether the cross-source comparison or synthesis itself is valid or meaningful. This limitation means that reports can score well on citation precision even when they draw incorrect conclusions from combining multiple sources. A better evaluation framework should assess synthesis quality by checking whether comparative claims, trends identified across sources, or integrated conclusions are logically sound given the cited evidence, not just whether individual citation-claim pairs are accurate. □

Deliverables: Your code for Section 4.1 and Section 4.2, as a completed version of the skeleton, together with your output JSON in the predefined formats. The results and discussion above.

5 Shared tasks [10 points]

5.1 Summarizing your current progress

In Homework 1, we asked you to benchmark baseline performance on all three shared tasks using a variety of models and inference strategies. Throughout the previous sections of Homework 2, we asked you to try at least one other inference strategy on the same tasks, on a subset of the models from the original set (`Qwen/Qwen3-4B`, `Qwen/Qwen3-4B-Instruct-2507`, and `Qwen/Qwen3-1.7B`).

For each model+task combination you ran from HW 1, copy your reported scores for the single best method + settings.

<u>**Solution:**</u>

Model	Decoding Settings	Score
Graph		
`Qwen/Qwen3-4B`	default	0.993
`Qwen/Qwen3-4B-Instruct-2507`	default	0.993
`Qwen/Qwen3-1.7B`	default	0.993
MMLU		
`Qwen/Qwen3-4B`	temp 1.5	0.74
`Qwen/Qwen3-4B-Instruct-2507`	temp 0.25	0.80
`Qwen/Qwen3-1.7B`	default	0.57
InfoBench		
`Qwen/Qwen3-4B`	beam 3	0.774
`Qwen/Qwen3-4B-Instruct-2507`	beam 25	0.785
`Qwen/Qwen3-1.7B`	typical	0.741

□

Now, fill out the table below with the *single best method* so far for each task, across both Homework 1 and Homework 2. Note that, since your implementation or hyperparameters may vary slightly, we are not grading for a "correct" answer here; this is to help you keep track of where your best method so far stands on each dataset.

<u>**Solution:**</u>

Task	Model	Decoding Settings	Score
Graph	Qwen3-4B-Instruct-2507	default	0.993
MMLU	Deep Research Agent (gpt-4o-mini)	Self-refine	0.83
InfoBench	Pairwise Reward	best-of-n	0.789

□

5.2 Planning improvements

For each of the shared tasks, describe at least one thing that you hope will help improve performance over your current scores, other than tuning hyperparameters for a method that you have already tried.

These could be additional sampling algorithms discussed in lecture or researched on your own, combining specific strategies implemented for either of the homework assignments so far, or any other ideas you have. Write a 1-3 sentence description of your proposed methods for each of the tasks. You do *not* have to implement your ideas for this homework (though it may be beneficial to implement these ideas for the end-of-semester shared task).

Solution:

Graph: I believe there is a bug in the code from homework 1, which is leading to much higher accuracy, as described in the prior table. After fixing that, I believe the accuracy would still be low. Hence, I would like to try strategies such as A* search and BFS, as they are useful for generating programs for solving problems. In the same process, I would use an interpreter/compiler for the program as a tool to process the code generated and give the desired response. I would also be interested in exploring FUDGE decoding with self-refining loops (possibly stop the loop incase the model has generated the correct response at any step).

MMLU: I would try performing self-refine with enable_thinking setting on, and pass the LLM's chain of thought generations as input in the feedback loop for it to process and be more thorough about its analysis.

InfoBench: I would implement a hybrid approach combining the best-of-n generation with pairwise reward model ranking, then use the top-k candidates as context for a final refinement pass that synthesizes and improves the best outputs. Additionally, I would explore using the deep research agent to gather supporting evidence from web searches before generating responses, similar to what achieved 0.83 on MMLU. □

Deliverables: The tables and written responses above. No code is required for this problem.

References

[1] Ahmad Beirami, Alekh Agarwal, Jonathan Berant, Alexander D'Amour, Jacob Eisenstein, Chirag Nagpal, and Ananda Theertha Suresh. Theoretical guarantees on the best-of-n alignment policy, 2025.

[2] Peter J. Bickel and Kjell A. Doksum. *Mathematical Statistic: Basic Ideas and Selected Topics*. Holden-Day Inc., Oakland, CA, 1977.

[3] Jonathan Bragg, Mike D'Arcy, Nishant Balepur, Dan Bareket, Bhavana Dalvi, Sergey Feldman, Dany Haddad, Jena D. Hwang, Peter Jansen, Varsha Kishore, Bodhisattwa Prasad Majumder, Aakanksha Naik, Sigal Rahamimov, Kyle Richardson, Amanpreet Singh, Harshit Surana, Aryeh Tiktinsky, Rosni Vasu, Guy Wiener, et al. Astabench: Rigorous benchmarking of ai agents with a holistic scientific research suite. *arXiv preprint*, 2025.

[4] Google DeepMind. Gemini Deep Research, 2025.

[5] Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. Deepresearch bench: A comprehensive benchmark for deep research agents. *arXiv preprint*, 2025.

[6] Bryan Eikema and Wilker Aziz. Is map decoding all you need? the inadequacy of the mode in neural machine translation. In *Proceedings of the 28th International Conference on Computational Linguistics*, Barcelona, Spain, December 2020. Association for Computational Linguistics.

[7] Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. Enabling large language models to generate text with citations. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

[8] Yuki Ichihara, Yuu Jinnai, Tetsuro Morimura, Kaito Ariu, Kenshi Abe, Mitsuki Sakamoto, and Eiji Uchibe. Evaluation of best-of-n sampling strategies for language model alignment, 2025.

[9] Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion, 2023.

[10] Chris Yuhao Liu, Liang Zeng, Jiacai Liu, Rui Yan, Jujie He, Chaojie Wang, Shuicheng Yan, Yang Liu, and Yahui Zhou. Skywork-reward: Bag of tricks for reward modeling in llms, 2024.

[11] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023.

[12] OpenAI. Introducing deep research, 2025.

[13] Liana Patel, Negar Arabzadeh, Harshit Gupta, Ankita Sundar, Ion Stoica, Matei Zaharia, and Carlos Guestrin. Deepscholar-bench: A live benchmark and automated evaluation for generative research synthesis. 2025.

[14] Perplexity. Introducing perplexity deep research, 2025.

[15] Amanpreet Singh, Joseph Chee Chang, Chloe Anastasiades, Dany Haddad, Aakanksha Naik, Amber Tanaka, Angele Zamarron, Cecile Nguyen, Jena D Hwang, Jason Dunkleberger, et al. Ai2 scholar qa: Organized literature synthesis with attribution. *arXiv preprint*, 2025.

[16] Li S. Yifei, Allen Chang, Chaitanya Malaviya, and Mark Yatskar. ResearchQA: Evaluating scholarly question answering at scale across 75 fields with survey-mined questions and rubrics. *arXiv preprint*, 2025.

[17] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert, 2020.