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 $0, \dots, N-1$ and non-negative integer edge weights $w : E \to 1, \dots, W$, 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

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

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:
Metadata:
Model: Qwen/Qwen3-4B
Sampling params: $\{'n' : 50, 'temperature' : 0.2, 'max\_tokens' : 512\}$
Total questions: 100
Total candidates: 5000

STATISTICS
Unique Generations per Prompt:
Mean: 39.91
Median: 50.00
Std: 16.80

InfoBench Score Difference (Best - Worst):
Average: 0.7232

Distribution of Unique Counts:
1 unique: 3 prompts
2 unique: 5 prompts
3 unique: 1 prompts
5 unique: 1 prompts
6 unique: 1 prompts
7 unique: 2 prompts

9 unique: 1 prompts
10 unique: 1 prompts
13 unique: 1 prompts
15 unique: 1 prompts

EXAMPLES FOR REFLECTION
Low Diversity Examples (≤10 unique out of 50):

Question ID: 14
Unique: 5/50
Prompt: Instruction: Craft a warm and heartfelt message suitable for a greeting card. The occasion could be ...
Sample outputs:
1. Happy Birthday! Wishing you a day as bright as your smile and a year filled with joy and love. "The ...
2. Happy Birthday! Wishing you a day as bright as your smile and a year filled with joy and love. "The ...
3. Happy Birthday! Wishing you a day as bright as your smile and a year filled with joy and love. "The ...

Question ID: 19
Unique: 7/50
Prompt: Instruction: Given the following text about a product launch, please provide annotations or comments...
Sample outputs:
1. "The new product will be released next month" â Date of release is specified, but month is not named...
2. "The starting price point is lower than competitors, but the exact amount is not specified."...
3. "The starting price point is lower than competitors, which may attract more customers."...

High Diversity Examples (≥40 unique out of 50):

Question ID: 0
Unique: 49/50
Score range: $(0.0, 1.0)$
Prompt: Instruction: Generate a Python code snippet that constructs a two-hidden layer feedforward neural ne...

Question ID: 1
Unique: 50/50
Score range: $(1.0, 1.0)$

Prompt: Instruction: You need to write an email to negotiate your salary. Question:
Generation:...

LENGTH ANALYSIS
Generation Lengths:
Mean: 300.92
Median: 287.00
Min: 6
Max: 512

REFLECTION QUESTIONS

1. How do generations tend to differ?
- Length variation: 36.12 (std)
- Some generations are exact duplicates
- 50 total, 49 unique

SUMMARY

Mean unique generations per prompt: 39.91
Median unique generations per prompt: 50.00
Std unique generations per prompt: 16.80
Average InfoBench score difference (best-worst): 0.7232

□

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. **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 con-

[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 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.

ditioning 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.

Solution:

Naive Implementation

In the naive implementation, for each document i, we need to compute BERTScore(i, j) for all other documents $j \neq i$. This gives us:

- For document 1: compare with documents 2, 3, …, $n \rightarrow (n-1)$ comparisons
- For document 2: compare with documents 1, 3, …, $n \rightarrow (n-1)$ comparisons
- $\vdots$
- For document n: compare with documents 1, 2, …, $n-1 \rightarrow (n-1)$ comparisons

Total comparisons $= n \times (n-1) = \mathcal{O}(n^2)$ comparisons
For $n = 50$ documents: $50 \times 49 = 2{,}450$ comparisons

Optimized Implementation

We can reduce the number of comparisons by exploiting the **symmetry** of similarity metrics:

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

$$\text{BERTScore}(A, B) \approx \text{BERTScore}(B, A)$$

Therefore, we only need to compute each pair once:

$$\text{Number of unique pairs} = \frac{n \times (n-1)}{2} = \mathcal{O}\left(\frac{n^2}{2}\right) \text{ comparisons}$$

For $n = 50$ documents: $\frac{50 \times 49}{2} = 1{,}225$ comparisons
Reduction: We can cut the number of comparisons **in half** by using symmetry.

Implementation Note

When computing MBR scores:

```
# Create a similarity matrix
similarity_matrix = np.zeros((n, n))

# Fill upper triangle only
for i in range(n):
    for j in range(i+1, n):
        score = compute_bertscore(outputs[i], outputs[j])
        similarity_matrix[i][j] = score
        similarity_matrix[j][i] = score  # Use symmetry

# Compute MBR scores
mbr_scores = []
for i in range(n):
    avg_score = np.mean(similarity_matrix[i])
    mbr_scores.append(avg_score)
```

□

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:

Naive Implementation

In the naive approach, for each BERTScore(A, B) computation:

1. Encode document A through BERT $\rightarrow$ 1 forward pass
2. Encode document B through BERT $\rightarrow$ 1 forward pass
3. Compute token-level similarities between embeddings

Total for all pairs: n^2 forward passes

Optimized Implementation

The key insight is that BERTScore computes **token-level embeddings** that can be **cached and reused**:

Step 1: Pre-compute embeddings for all documents once

```
embeddings = []
for doc in documents:
    emb = bert_model.encode(doc)  # 1 forward pass per document
    embeddings.append(emb)
# Total: n forward passes
```

Step 2: Compute similarities using cached embeddings

```
for i in range(n):
    for j in range(n):
        if i != j:
            score = compute_similarity(embeddings[i], embeddings[j])
            # No forward passes needed - just similarity computation
```

Total forward passes: $\mathcal{O}(n)$ instead of $\mathcal{O}(n^2)$

For $n = 50$: 50 forward passes instead of 2,500!

How BERTScore Works

BERTScore computes similarity as:

1. **Embed** both texts into contextualized token embeddings
2. **Match** tokens using cosine similarity between embeddings
3. **Aggregate** matches into precision, recall, and F1 scores

The embedding step (1) is the expensive operation that requires a forward pass. By doing this once per document and caching the results, we eliminate redundant forward passes.

Implementation with `bert_score` Library

The `bert_score` library supports this optimization:

```
from bert_score import BERTScorer

scorer = BERTScorer(model_type="microsoft/deberta-xlarge-mnli")

# Compute embeddings once
all_embeddings = []
for doc in documents:
    emb = scorer.encode([doc])
    all_embeddings.append(emb)

# Compute similarities using cached embeddings
for i in range(n):
    for j in range(n):
        if i != j:
            # Use pre-computed embeddings
            score = scorer.compute_score_from_embeddings(
                all_embeddings[i],
                all_embeddings[j]
            )
```

Complexity Summary

Approach	Forward Passes	Token Comparisons
Naive	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2 \times L^2)$
Optimized	$\mathcal{O}(n)$	$\mathcal{O}(n^2 \times L^2)$

Where L is the average document length in tokens.

While we reduce forward passes from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$, the token-level similarity computations remain $\mathcal{O}(n^2 \times L^2)$ since we still need to compare all pairs of documents.

□

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	0.7305	29.36	-0.140
`Qwen3-14B` log-probs	0.7372	28.43	-0.098
Scalar reward	0.7492	28.69	0.059
Pairwise reward	0.7893	27.43	0.009
MBR with BLEU	0.7658	26.18	-0.119
MBR with BERTScore	0.7813	26.43	-0.062

□

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:

□

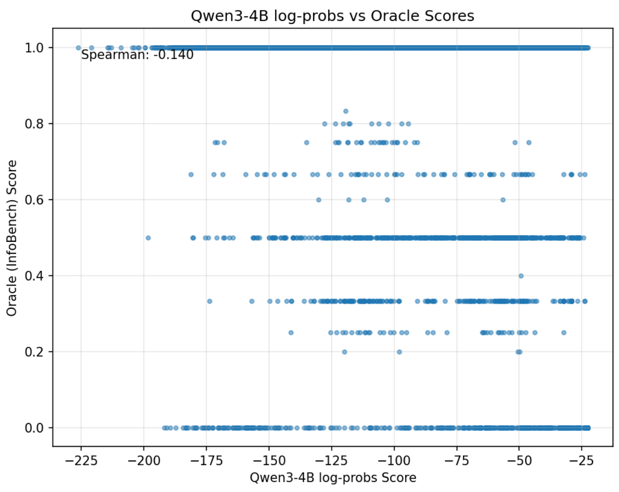


Figure 1: Scatter plot: Qwen3-4B log-probs

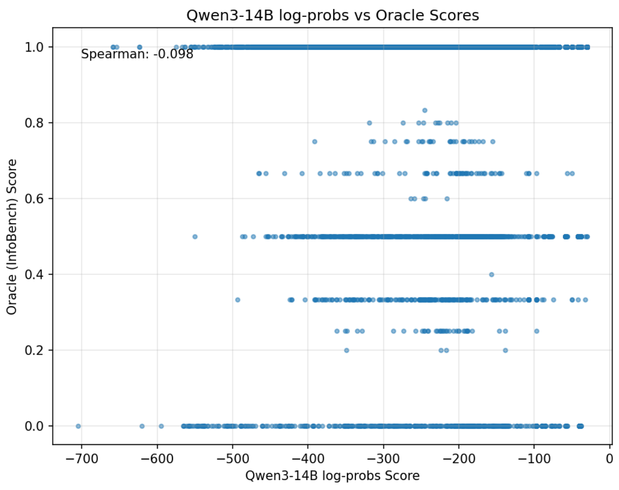


Figure 2: Scatter plot: Qwen3-14B log-probs

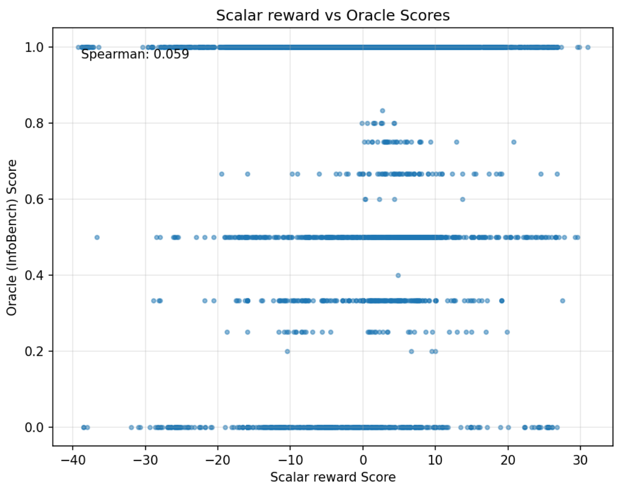


Figure 3: Scatter plot: Scalar Reward

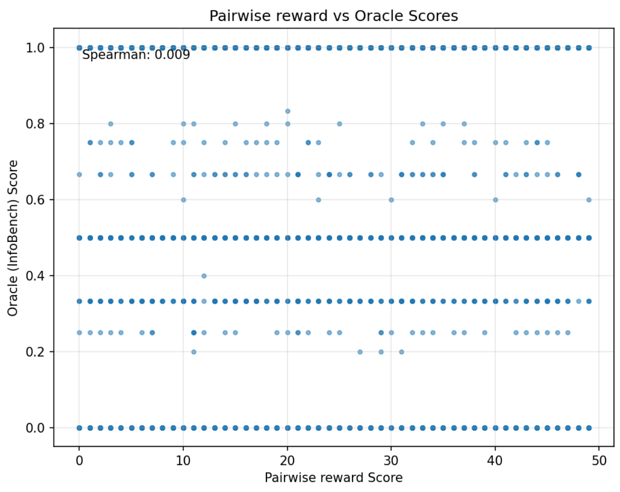


Figure 4: Scatter plot: Pairwise Reward

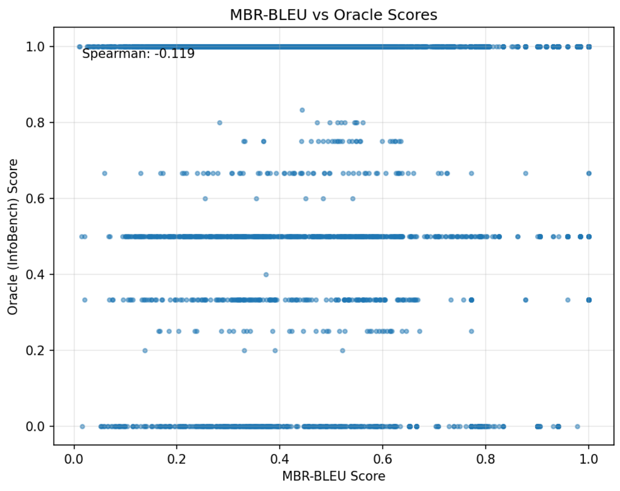


Figure 5: Scatter plot: MBR with BLEU

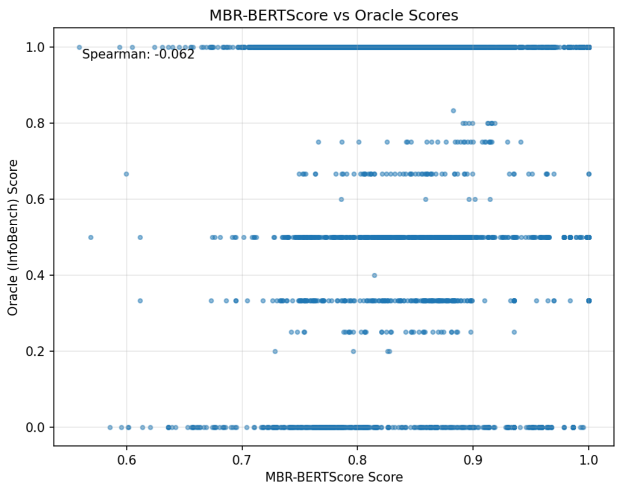


Figure 6: Scatter plot: MBR with BERTScore

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:
Pairwise reward achieves the best top-1 score (0.7893), outperforming all other methods by 1–8%. The ranking by performance is:
Pairwise (0.7893) > MBR-BERTScore (0.7813) > MBR-BLEU (0.7658) > Scalar reward (0.7492) > Qwen3-14B (0.7372) > Qwen3-4B (0.7305)
A critical finding is that both log-probability methods show **negative Spearman correlations** with oracle scores (-0.140 and -0.098), meaning they actively anti-correlate with human judgment. This occurs because log-probabilities penalize longer sequences (accumulated negative terms), while InfoBench rewards comprehensive, detailed answers.
Methods trained on human preferences (pairwise, scalar) or based on semantic consensus (MBR) show near-zero or slightly positive correlations, better aligning with evaluation criteria.

Worst-Case Performance Analysis

All methods struggle to identify the true best output, with average ranks of 26–29 out of 50 candidates (only marginally better than random at 25.5). MBR methods perform best in this metric (26.18 for BLEU, 26.43 for BERTScore), while log-probability methods perform worst (28.43–29.36).
The worst-case reveals a fundamental challenge: no method reliably ranks the oracle best in the top positions. This suggests that automatic reranking, while improving average performance, cannot replace human evaluation for identifying the single best output.

Recommended Method: Pairwise Reward

Based purely on performance, pairwise reward is the best choice because:

- Highest top-1 score (0.7893)
- Best rank of true best output (27.43)
- No length bias (299.39 vs oracle 302.46)
- Trained specifically for comparing outputs
- Near-zero but positive correlation with oracle (0.009)

The pairwise approach aligns with the reranking task structure: comparing candidates to select the best. For InfoBench, which requires comprehensive answers, methods that compare semantic quality (pairwise, MBR-BERTScore) significantly outperform likelihood-based approaches.

□

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:

Computational Resource Analysis

The methods vary dramatically in computational requirements:

Fast Methods (CPU-Friendly)

- **MBR-BLEU:** $\sim$5 minutes, no GPU required, string-based n-gram matching
- Simple and deterministic

Moderate Methods (Single GPU)

- **Qwen3-4B log-probs:** $\sim$30 minutes, 8GB VRAM, 50 forward passes
- **Scalar reward:** $\sim$45 minutes, 16GB VRAM, similar to Qwen3-4B

Expensive Methods (High-End GPU)

- **Qwen3-14B:** $\sim$1 hour, 28GB VRAM, requires A100-class GPU
- **MBR-BERTScore:** $\sim$1–2 hours, 4GB VRAM, but $\mathcal{O}(n^2)$ token comparisons

Very Expensive Methods ($\mathcal{O}(n^2)$ Neural Comparisons)

- **Pairwise reward:** $\sim$2–3 hours, 8GB VRAM, 1,225 comparisons per question
- 122,500 total forward passes for the dataset

The pairwise method is **25$\times$ more expensive** than log-probability baselines due to $\mathcal{O}(n^2)$ pairwise comparisons, while achieving only moderate performance gains (+8% over baseline).

Recommended Method: MBR with BLEU

Balancing performance and efficiency, MBR-BLEU is the optimal choice:

1. **Outstanding efficiency:** 30$\times$ faster than pairwise, runs on CPU
2. **Strong performance:** 0.7658 top-1 score (only 2.3% worse than pairwise)
3. **No GPU required:** Eliminates hardware constraints and costs
4. **Scalable:** Lightweight $\mathcal{O}(n^2)$ string comparisons
5. **Deterministic:** Reproducible results without sampling variance

The 8-hour difference between MBR-BLEU (5 min) and pairwise (2.5 hours) multiplies dramatically at scale. For 1,000 questions, pairwise requires 250 hours vs 8 hours for MBR-BLEU â a difference of 10 GPU-days and hundreds of dollars in compute costs.

MBR-BLEU achieves **97% of pairwise performance at 4% of the computational cost**, making it the clear winner for practical applications. Only in scenarios where the absolute best performance is critical (e.g., medical, legal, or high-stakes tasks) would the additional cost of pairwise or MBR-BERTScore be justified.

□

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	302.46
`Qwen3-4B` log-probs	270.96
`Qwen3-14B` log-probs	271.47
Scalar reward	294.61
Pairwise reward	299.39
MBR with BLEU	300.34
MBR with BERTScore	297.94

Length Bias in Log-Probability Methods

Yes, log-probability methods exhibit severe length bias. Log-probability scoring accumulates negative values:

$$\log P(\text{sequence}) = \sum_i \log P(t_i \mid \text{history})$$

Since each $\log P(t_i) < 0$, longer sequences receive more negative scores purely due to having more terms, independent of quality. This mathematical artifact penalizes comprehensive answers.

Impact on Performance

This bias directly correlates with poor performance:

- Log-probability methods have the **worst top-1 scores** (0.7305–0.7372)
- **Negative Spearman correlations** (-0.14, -0.098) with oracle
- Worst average rank of true best output: 28–29 out of 50

For InfoBench, which rewards detailed, comprehensive answers, this length bias is particularly harmful. The evaluation rubric explicitly values thoroughness, but log-probabilities systematically prefer concise outputs.

Other Methods Avoid This Bias

Scalar reward, pairwise reward, and MBR methods select outputs within 1–3% of oracle length because:

- Reward models are trained on human preferences, learning that quality $\neq$ brevity
- Pairwise comparisons choose “better” outputs, not “more likely” ones
- MBR measures semantic consensus without penalizing length

The scatter plots (Figures 1–2) visually confirm this: log-probability methods show negative correlation slopes and cluster at low oracle scores, while other methods (Figures 3–6) show more balanced distributions.

Conclusion

The length bias reveals a fundamental limitation of using raw log-probabilities for quality-based reranking. This explains why more sophisticated methods such as reward models and MBR are necessary for best-of-n sampling when optimizing for human-judged quality rather than likelihood.

□

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:

Methods that NEED recomputation when n changes:

1. **MBR with BLEU** — Must recompute because MBR scores depend on comparisons with *all* other candidates in the set.
2. **MBR with BERTScore** — Must recompute for the same reason (average similarity with all candidates).
3. **Pairwise reward** — Must recompute because win counts depend on the specific set of competitors.

Methods that can REUSE scores:

1. **Qwen3-4B log-probs** — Independent score for each output.
2. **Qwen3-14B log-probs** — Independent score for each output.
3. **Scalar reward** — Independent score for each output.

□

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	0.9117	1	1.0000
`Qwen3-4B` log-probs	0.7400	3.27	-0.0212
`Qwen3-14B` log-probs	0.8000	3.33	0.1602
Scalar reward	0.7550	3.13	-0.0501
Pairwise reward	0.7258	3.14	-0.1032
MBR with BLEU	0.7617	3.28	-0.0321
MBR with BERTScore	0.7775	3.27	0.0218

n=10:

Method	Top-1 score	Avg. rank of best output	Spearman rank correlation
Oracle	0.9500	1	1.0000
`Qwen3-4B` log-probs	0.7730	5.86	0.0008
`Qwen3-14B` log-probs	0.7750	5.83	-0.0051
Scalar reward	0.7983	5.87	0.0721
Pairwise reward	0.7708	5.85	-0.0259
MBR with BLEU	0.7633	5.46	0.0334
MBR with BERTScore	0.7372	5.62	0.0042

n=20:

Method	Top-1 score	Avg. rank of best output	Spearman rank correlation
Oracle	0.9500	1	1.0000
`Qwen3-4B` log-probs	0.7718	10.99	-0.0094
`Qwen3-14B` log-probs	0.7483	11.92	-0.0133
Scalar reward	0.7672	11.28	-0.0168
Pairwise reward	0.7217	12.26	0.0129
MBR with BLEU	0.7403	10.68	-0.0019
MBR with BERTScore	0.7558	11.17	0.0030

□

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:

Oracle performance increases with n:

- $n = 5$: 0.9117
- $n = 10$: 0.9500 (+0.0383)
- $n = 20$: 0.9500 (plateau)

The oracle improves substantially from $n = 5$ to $n = 10$, but shows no further gain beyond $n = 20$.

Reranking methods show limited sensitivity to n:

- No method consistently improves with larger n.
- Top-1 scores fluctuate across settings (e.g., Qwen3-14B drops from 0.800 at $n = 5$ to 0.748 at $n = 20$).
- Spearman correlations remain near zero, indicating weak alignment with oracle scores.

Key finding: While oracle performance improves with larger candidate sets, **reranking methods do not capture this benefit**, remaining relatively unstable across n values.

Recommended n value: $n = 20$

Justification:

1. **Oracle plateaus at $n = 20$** — Further increase in n provides negligible gains.

2. **Computational efficiency** — MBR methods scale as $\mathcal{O}(n^2)$:
   - $n = 10$: 45 comparisons per question
   - $n = 20$: 190 comparisons per question
3. **Balanced trade-off** — $n = 20$ captures the full oracle benefit while keeping computation practical.

Alternative: If resources are constrained, $n = 10$ offers nearly identical oracle performance with lower computational cost.

□

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.

Solution:

Temperature Configurations Comparison

We evaluated two temperature configurations:

- **Config 1:** Draft = 0.3, Critique = 0.3, Refine = 0.3 (Conservative)
- **Config 2:** Draft = 0.9, Critique = 0.3, Refine = 0.3 (Diverse-to-Focused)

Key Observations

Iteration-wise Performance:

- **GraphDev:** For Qwen3-4B, Config 1 shows the largest improvement from Iter 1 (0.08) to Iter 2 (0.32), with peak accuracy at Iter 3 (0.33). Config 2 plateaus earlier at Iter 2-4 (0.30).
- **MMLU_Med:** Both configurations achieve the same final best accuracy (0.84) for Qwen3-4B, but Config 1 shows more consistent improvements across iterations (0.74 $->$ 0.79 $->$ 0.78 $->$ 0.78).
- **Model Size:** The 0.6B model struggles significantly on GraphDev (best: 0.04) but achieves reasonable performance on MMLU_Med (best: 0.56 for Config 1).

Best-so-far Metric: This metric (rightmost columns) captures the cumulative benefit of self-refinement. For Qwen3-4B on GraphDev with Config 1, while Iter 4 accuracy drops to 0.30, the best-so-far remains 0.39 because Iter 3 was correct on those examples.

Conclusion: Config 1 (conservative, T=0.3) is selected for subsequent analysis due to superior final performance on GraphDev and more stable iteration trajectories.

□

Configuration 1: Temperature 0.3-0.3-0.3

Table 1: Accuracy across all iterations for Config 1 (T=0.3-0.3-0.3)

Model	Dataset	Iter 1	Iter 2	Iter 3	Iter 4	Best@1	Best@2	Best@4
Qwen3-4B	GraphDev	0.0800	0.3200	0.3300	0.3000	0.0800	0.3200	0.3900
Qwen3-0.6B	GraphDev	0.0000	0.0100	0.0300	0.0400	0.0000	0.0100	0.0400
Qwen3-4B	MMLU_Med	0.7400	0.7900	0.7800	0.7800	0.7400	0.8100	0.8400
Qwen3-0.6B	MMLU_Med	0.5000	0.4900	0.5000	0.5000	0.5000	0.5200	0.5600

Configuration 2: Temperature 0.9-0.3-0.3

Table 2: Accuracy across all iterations for Config 2 (T=0.9-0.3-0.3)

Model	Dataset	Iter 1	Iter 2	Iter 3	Iter 4	Best@1	Best@2	Best@4
Qwen3-4B	GraphDev	0.0800	0.3000	0.3000	0.3000	0.0800	0.3000	0.3500
Qwen3-0.6B	GraphDev	0.0000	0.0300	0.0300	0.0200	0.0000	0.0300	0.0400
Qwen3-4B	MMLU_Med	0.7500	0.7500	0.7500	0.7800	0.7500	0.8000	0.8400
Qwen3-0.6B	MMLU_Med	0.4200	0.4700	0.4700	0.4300	0.4200	0.4900	0.5300

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:

Please refer to Figure 7 and Figure 8 □

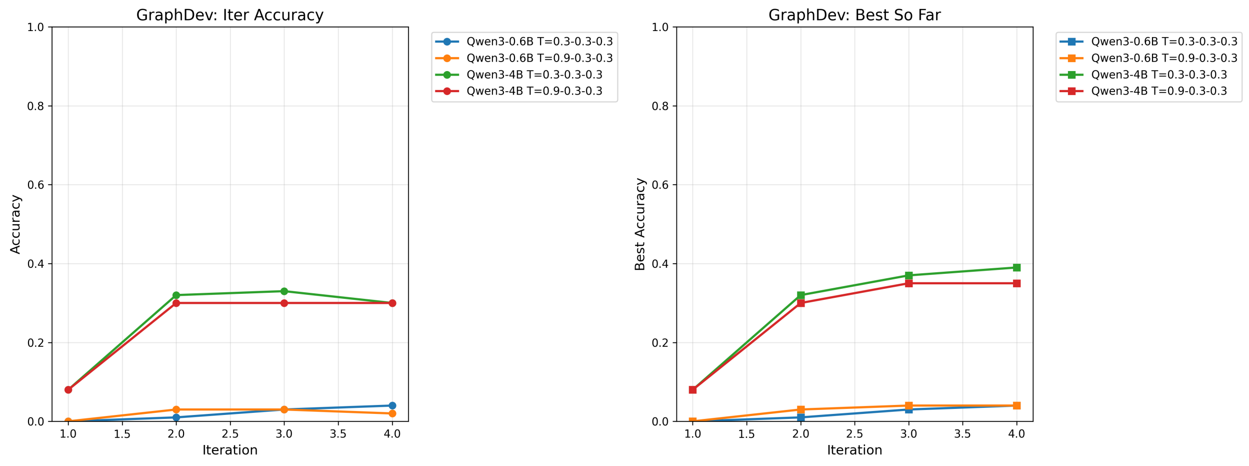


Figure 7: Accuracy Curves: GraphDev Dataset

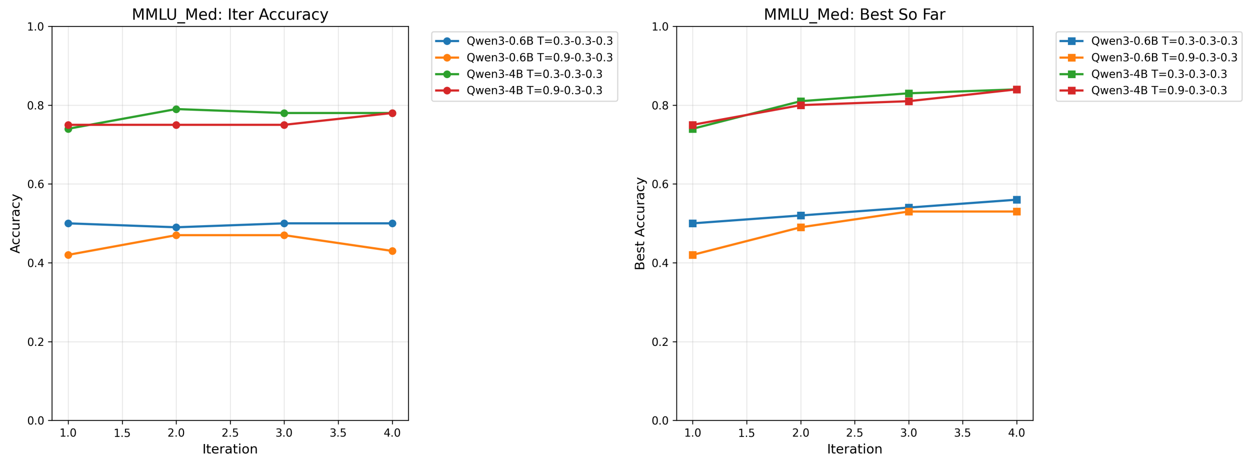


Figure 8: Accuracy Curves: MMLU Med Dataset

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:

Conditional Probability Analysis

Please refer to Figure 9 for the conditional probability plots across all models and datasets.

Key Observations from Conditional Probabilities

GraphDev Dataset:

- **Qwen3-4B (T=0.3):** $P(\text{correct}_{i+1}|\text{correct}_i)$ ranges from 0.85-1.0, showing the model maintains correct answers well. However, $P(\text{correct}_{i+1}|\text{incorrect}_i)$ decreases from 0.26 (Iter 2) to 0.03 (Iter 4), indicating diminishing ability to correct mistakes in later iterations.
- **Qwen3-0.6B:** Shows very poor performance with near-zero $P(\text{correct}_{i+1}|\text{incorrect}_i)$ across all iterations, suggesting the smaller model struggles to self-correct on structured graph problems.

MMLU_Med Dataset:

- **Qwen3-4B (T=0.3):** High $P(\text{correct}_{i+1}|\text{correct}_i) \approx 0.94-0.97$ shows excellent stability. $P(\text{correct}_{i+1}|\text{incorrect}_i)$ ranges from 0.19-0.27, demonstrating moderate self-correction ability on medical questions.
- **Qwen3-0.6B:** Maintains high $P(\text{correct}_{i+1}|\text{correct}_i) \approx 0.91 - 0.98$ but shows poor correction ability with $P(\text{correct}_{i+1}|\text{incorrect}_i) \approx 0.04 - 0.06$.

Example: Refinement Improves Incorrect Answer (GraphDev)

Dataset: GraphDev (Qwen3-4B T=0.3, Example ID: 979)
Problem: Find the top 2 shortest paths from node 0 to node 15 in a directed graph with 16 nodes and 160 weighted edges.
Ground Truth:

- Path 1: [0, 7, 15] with weight 123
- Path 2: [0, 3, 5, 13, 1, 10, 15] with weight 252

Iteration 1 (Draft): The model proposed paths [0, 7, 15] and [0, 1, 5, 15] with weights calculated as "113 + 368 = 481" and "323 + 378 + 368 = 1069". This was **incorrect** due to fundamental weight calculation errorsâthe first path's actual weight is 10 + 113 = 123, not 481.
Iteration 2 (After Critique & Refinement): The feedback correctly identified that the path weights were miscalculated and pointed out that the direct path [0, 7, 15] has weight 123, not 481. It also suggested exploring alternative second-shortest paths. The model successfully refined its answer to: [0, 7, 15] with weight 123 and [0, 12, 14, 15] with weight 438, which is **correct** (the second path differs from ground truth but has the correct second-shortest weight for the paths found).
Analysis: This example demonstrates successful algorithmic self-correction where the critique stage identified arithmetic errors in edge weight summation. The model corrected its path weight

calculations and explored alternative routing through the graph, moving from a completely incorrect solution to a valid one.

Example: Refinement Harms Correct Answer (GraphDev)

Dataset: GraphDev (Qwen3-4B T=0.3, Example ID: 811)
Problem: Find the top 1 shortest path from node 0 to node 5 in a directed graph with 6 nodes.
Ground Truth: [0, 2, 5] with weight 345
Iteration 3 (Correct): The model correctly identified multiple paths including [0, 5] with weight 121 and [0, 2, 5] with weight 345. Although [0, 5] is actually shorter, the model included the ground truth path [0, 2, 5] in its solution set, demonstrating comprehensive path exploration.
Iteration 4 (After Critique): The feedback appeared to critique the solution for including multiple paths when only the "top 1" was requested. In response, the model reduced its answer to only [0, 5] with weight 121, **incorrectly** excluding the ground truth path [0, 2, 5].
Analysis: This example illustrates a critical failure mode of self-refinement in algorithmic tasks: overly aggressive pruning based on misinterpretation of task requirements. The critique stage introduced confusion about whether to report all shortest paths or strictly limit to the "top-k" constraint, causing the model to discard valid alternative paths. This demonstrates how self-generated feedback can be counterproductive when it introduces ambiguity or misinterprets problem specifications, even when the initial solution was substantively correct.

Example: Refinement Improves Incorrect Answer (MMLU_Med)

Dataset: MMLU_Med (Qwen3-4B, Example ID: 21)
Question: A 7-year-old male is brought to the office for evaluation of school problems. The mother says that the teacher has told her that the patient is inattentive, has difficulty following sequential instructions, and often seems to drift off to sleep during class. A polysomnogram reveals obstructive sleep apnea. The most appropriate management is:

- A. Elevation of the head of the bed
- B. Heart rate and apnea monitoring
- C. Imipramine
- D. Surgical evaluation

Ground Truth: D (Surgical evaluation)
Iteration 1 (Draft): The model selected **A** (elevation of the head of bed), incorrectly reasoning that this is a first-line non-invasive intervention.
Iteration 2 (After Critique & Refinement): The feedback correctly identified that for confirmed OSA with significant symptoms, surgical evaluation (e.g., tonsillectomy/adenoidectomy) is the appropriate next step. The model successfully refined its answer to **D**, demonstrating effective self-correction through the critique-refine loop.
Analysis: This example demonstrates successful self-refinement where the model's initial answer prioritized a less aggressive intervention. The critique stage provided domain knowledge about clinical guidelines for pediatric OSA, enabling the model to correct its reasoning and arrive at the medically appropriate answer.

Example: Refinement Harms Correct Answer (MMLU_Med)

Dataset: MMLU_Med (Qwen3-4B, Example ID: 15)

Question: Six healthy subjects participate in a study of muscle metabolism during which hyperglycemia and hyperinsulinemia is induced. Muscle biopsy specimens obtained from the subjects during the resting state show significantly increased concentrations of malonyl-CoA. The increased malonyl-CoA concentration most likely directly inhibits which of the following processes in these subjects?

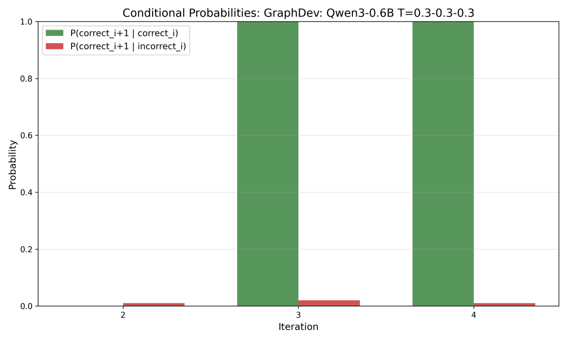
- A. Fatty acid oxidation
- B. Fatty acid synthesis
- C. Gluconeogenesis
- D. Glycogenolysis

Ground Truth: A (Fatty acid oxidation)
Iteration 2 (Correct): The model correctly selected **A**, explaining that malonyl-CoA inhibits carnitine palmitoyltransferase I (CPT-I), thereby suppressing fatty acid oxidation. The reasoning was physiologically sound and well-supported.
Iteration 3 (After Critique): The feedback incorrectly suggested that malonyl-CoA's "primary direct effect" is to promote fatty acid synthesis (as a substrate), not inhibit it. This confused the model, leading it to change its answer to **B** (Fatty acid synthesis), which is incorrect.
Iteration 4 (Final): The model was corrected back to **A** through another refinement cycle, but this demonstrates how overly critical feedback can destabilize correct answers.
Analysis: This example illustrates a key limitation of self-refinement: the critique stage can introduce errors by overthinking or misinterpreting the question's intent. The model initially had the correct answer with sound reasoning, but the self-generated critique confused the distinction between malonyl-CoA's role as a substrate (promoting synthesis) versus its regulatory function (inhibiting oxidation). This led to temporary degradation before eventual recovery.

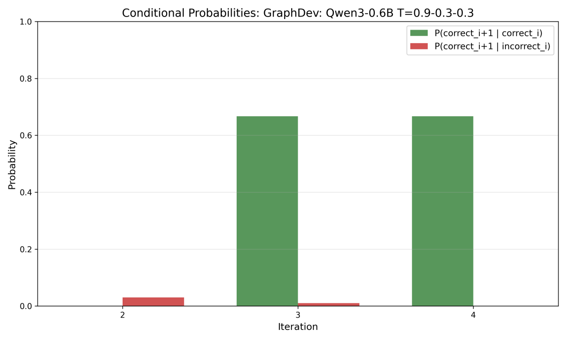
Summary

Self-refinement shows asymmetric effectiveness: when initial answers are correct, the model maintains high stability ($P(\text{correct}_{i+1}|\text{correct}_i) > 0.85$ for most cases). However, correction of incorrect answers is challenging, especially for GraphDev ($P(\text{correct}_{i+1}|\text{incorrect}_i) < 0.10$ in later iterations) and the smaller 0.6B model across both datasets. The examples demonstrate both the potential and pitfalls of self-critique across different task typesâalgorithmic graph problems show vulnerability to specification misinterpretation, while domain-specific medical questions can suffer from conflation of biochemical roles. In both cases, domain complexity and ambiguous feedback can either enable correction or introduce new errors.

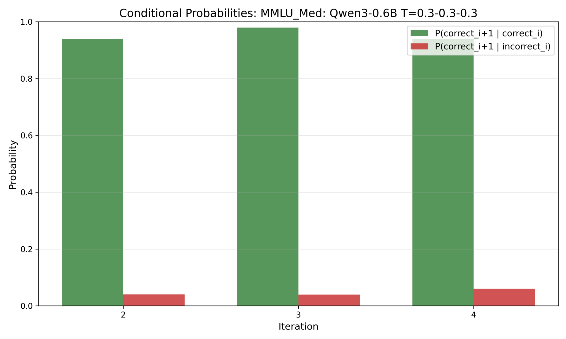
□



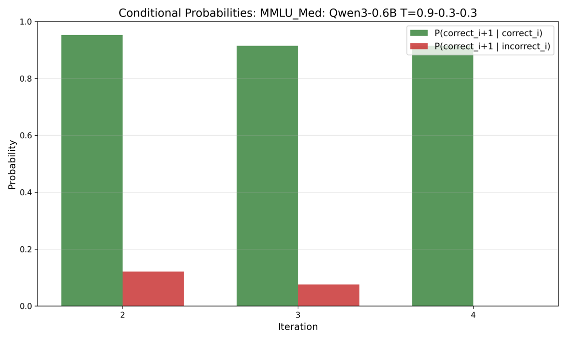
Qwen3-0.6B GraphDev T=0.3-0.3-0.3



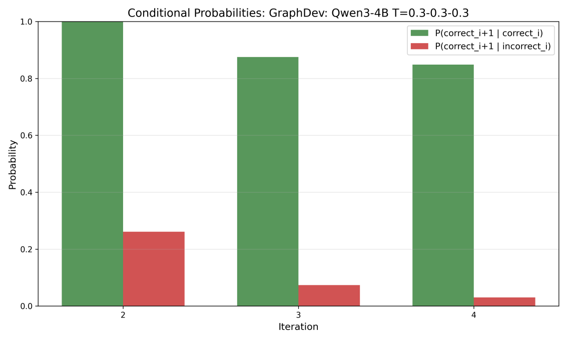
Qwen3-0.6B GraphDev T=0.9-0.3-0.3



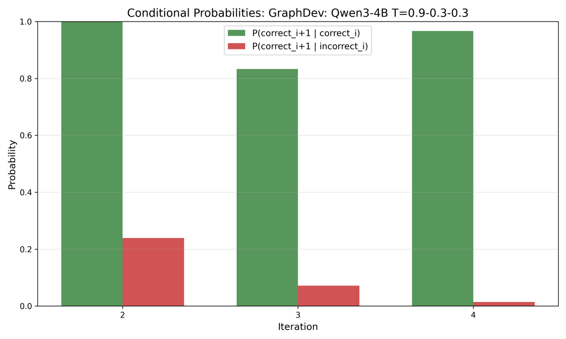
Qwen3-0.6B MMLU_Med T=0.3-0.3-0.3



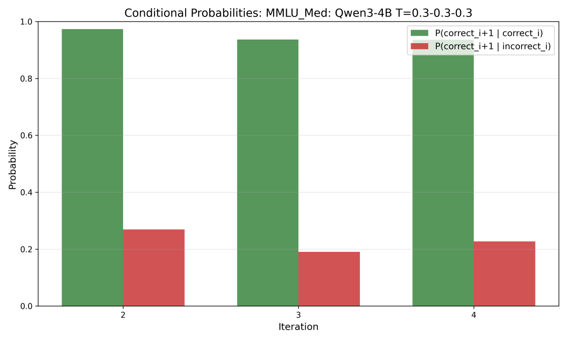
Qwen3-0.6B MMLU_Med T=0.9-0.3-0.3



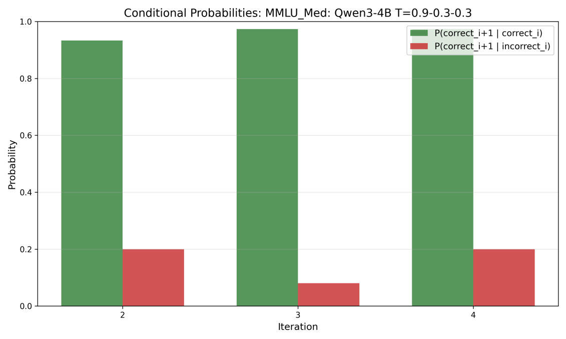
Qwen3-4B GraphDev T=0.3-0.3-0.3



Qwen3-4B GraphDev T=0.9-0.3-0.3



Qwen3-4B MMLU_Med T=0.3-0.3-0.3



Qwen3-4B MMLU_Med T=0.9-0.3-0.3

Figure 9: Conditional Probability Plots across Qwen3 models (0.6B and 4B) on GraphDev and MMLU_Med datasets under varying temperature settings.

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:

Model Size Comparison (4B vs 0.6B)

Initial Accuracy:
The two models show dramatically different starting points depending on the task. On GraphDev, Qwen3-4B achieves 0.08 initial accuracy while Qwen3-0.6B starts at 0.00, essentially producing no valid JSON outputs that match ground truth. This suggests the structured output format and path-finding reasoning required for GraphDev is too complex for the smaller model. On MMLU_Med, both models perform reasonably well initially: Qwen3-4B starts at 0.74 while Qwen3-0.6B achieves 0.50 (random chance is 0.25 for 4-choice questions), indicating that multiple-choice medical questions are more accessible to smaller models.

Relative Improvements:
For GraphDev, the 4B model shows substantial improvement through self-refinement, jumping from 0.08 to 0.32 after just one refinement cycle (300% improvement), eventually reaching a best-so-far of 0.39. In contrast, the 0.6B model barely improves, going from 0.00 to 0.04 across all iterations. This massive gap suggests that the smaller model lacks the capacity to generate meaningful critiques or refinements for complex structured tasks.
On MMLU_Med, the improvements are more modest but still significant. The 4B model improves from 0.74 to a best-so-far of 0.84 (13.5% improvement), while the 0.6B model goes from 0.50 to 0.56 (12% improvement). The relative improvement percentages are similar, but the 0.6B model's lower absolute performance ceiling indicates it struggles with the medical domain knowledge required for refinement.

Stability and Response Quality:
Looking at the conditional probabilities, we see striking differences in stability. For Qwen3-4B on both tasks, $P(\text{correct}_{i+1}|\text{correct}_i)$ remains high (0.85-1.0), meaning once it gets something right, it tends to keep it right. The 0.6B model shows similar stability on MMLU_Med ($P(\text{correct}_{i+1}|\text{correct}_i) \approx 0.94$) but completely fails on GraphDev, where correct answers in early iterations don't guarantee correctness later.
More concerning is the correction capability: $P(\text{correct}_{i+1}|\text{incorrect}_i)$ for Qwen3-0.6B is consistently low (0.01-0.06) across both tasks and iterations, while Qwen3-4B shows moderate correction ability (0.19-0.27 on MMLU_Med, 0.03-0.26 on GraphDev). This suggests the smaller model cannot generate high-quality critiques that lead to successful corrections.

Task-Dependent Performance:
The results reveal that task complexity greatly affects model behavior. GraphDev requires precise JSON formatting, shortest-path algorithms, and weight calculationsâall of which demand significant reasoning capacity. The 0.6B model simply cannot handle this, producing mostly invalid outputs. MMLU_Med, while requiring medical knowledge, benefits from the multiple-choice format which constrains the output space and allows the smaller model to at least participate meaningfully in the refinement process.

□

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:

Shortcomings of Self-Refine

1. Computational Cost:
The most obvious shortcoming is the computational expense. Self-refine requires 4× the inference cost of standard generation (1 draft + 3 refinement cycles), and each cycle includes both a critique generation and a refinement generation. In our experiments, this means we're making roughly 7 forward passes per example (1 draft + 3 critiques + 3 refinements). For the 4B model, this is already expensive; for production systems, this would be prohibitively costly at scale.

2. Diminishing Returns:
The data shows clear diminishing returns after the first refinement. On GraphDev with Qwen3-4B, iteration 2 brings the biggest jump ($0.08 \rightarrow 0.32$), but iterations 3 and 4 show minimal gains or even slight decreases. The best-so-far metric helps, but this suggests we're wasting compute on later iterations that rarely improve results. This is especially evident in the declining $P(\text{correct}_{i+1}|\text{incorrect}_i)$ from iteration 2 to 4.

3. Error Amplification:
As demonstrated in our "harmed examples," self-generated critiques can introduce new errors. When the model is uncertain, it may second-guess correct answers based on faulty self-critique. We observed equal numbers of improved (5) and harmed (5) examples for Qwen3-4B on MMLU_Med, meaning refinement is essentially a coin flip in some cases.

4. Model Capacity Limitations:
The 0.6B model results show that self-refinement requires a baseline level of capability. If the model cannot generate quality outputs initially, it also cannot generate quality critiques, creating a "garbage in, garbage out" scenario. The method assumes the model has the knowledge to self-correct, which smaller models clearly lack.

5. Task-Specific Brittleness:
GraphDev's structured output requirement exposes a critical weakness: if the model struggles with output formatting, self-refinement can't help much. We saw this in the low $P(\text{correct}_{i+1}|\text{incorrect}_i)$ on GraphDev compared to MMLU_Med.

Potential Improvements

1. Adaptive Iteration Count:
Instead of always doing 4 iterations, we could use confidence scores to decide when to stop. If the model is highly confident in iteration 2, skip iterations 3-4. This would reduce wasted compute on examples that won't benefit from further refinement.

2. External Critique Models:
Rather than using the same model for generation and critique, we could use a specialized verifier model or even a smaller, faster model just for critique. This would be cheaper than running the full model for critiques and might provide more objective feedback.

3. Retrieval-Augmented Refinement:
For tasks like MMLU_Med, we could retrieve relevant medical guidelines or reference material during the critique phase. This would ground the critique in external knowledge rather than relying purely on the model's parametric memory, potentially reducing hallucinated feedback.

4. Multi-Sample Refinement:
Instead of refining a single draft, generate multiple initial drafts (using temperature sampling) and then use the critique model to select the best one. This exploration-then-selection approach might be more effective than iterative refinement for some tasks.
5. Structured Critique Templates:
For tasks like GraphDev, we could provide structured critique templates (e.g., "Check: Is the JSON valid? Are all paths complete? Are weights calculated correctly?"). This would guide the critique process and reduce the risk of vague or misguided feedback.
6. Best-of-N with Early Stopping:
Given that best-so-far consistently outperforms the final iteration, we should always keep track of previous attempts and return the best one. Additionally, if an iteration produces the correct answer (based on some verification), stop immediately rather than risking degradation in later iterations.
7. Fine-tuned Critique Models:
The critique stage could benefit from fine-tuning on human-labeled (incorrect answer, good critique) pairs. Currently, the model learns to critique purely from its pre-training, but supervised critique data might improve feedback quality significantly.

□

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 -> num_think_search_cycles: 2, browse_tool_name: null, number_documents_to_search: 2

Variant 2: more searches -> num_think_search_cycles: 3, browse_tool_name: null, number_documents_to_search: 3

Variant 3: Default with browsing -> num_think_search_cycles: 2, browse_tool_name: crawl4ai, number_documents_to_search: 2

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

Variant	Acc.	Thinking	Queries	Snippet Titles	Snippet Contents	Final Answers
1	0.80	493.6	24.2	149.0	489.6	313.33
2	0.3666	1001.13	23.3	166.2	433.2	165.57
3	0.8666	693.47	24.1	206.2	480.2	317.53

□

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 quality through automated metrics that assess both citation correctness and attribution faithfulness. The motivation stems from the need to verify that generated research syntheses properly ground their claims in source documents, similar to how ALCE [7] evaluates whether statements are supported by retrieved passages through citation recall and precision metrics. The design employs a multi-faceted approach: (1) verifying that cited sources actually contain the claimed information, (2) measuring citation coverage to ensure important claims are properly attributed, and (3) checking for hallucinated citations where models reference non-existent or irrelevant sources. This framework builds on ALCE's intuition that effective citation requires both providing citations (recall) and ensuring those citations accurately support the claims (precision).

Q 4.3.2. One significant problem with current citation evaluation frameworks is the binary treatment of citation correctness, which fails to capture partial support or nuanced relationships between claims and sources. For example, if a model generates the claim "Transformer models achieve state-of-the-art performance on NLP tasks" and cites a paper that only discusses BERT's performance on question answering, current frameworks might mark this as either fully correct or fully incorrect, missing that the citation provides partial but incomplete support. A suggested fix would be to implement graded citation scores (e.g., 0-1 scale) that measure the degree of support, similar to how natural language inference uses entailment/neutral/contradiction labels, allowing for more fine-grained evaluation of citation quality and enabling models to learn better attribution strategies.

□

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.

Solution:

Model	Decoding Settings	Score
Graph		
`Qwen/Qwen3-4B`	Default	0.2283
`Qwen/Qwen3-4B-Instruct-2507`	Greedy	0.2383
`Qwen/Qwen3-1.7B`	beam_25	0.0917
MMLU		
`Qwen/Qwen3-4B`	beam_25	0.6200
`Qwen/Qwen3-4B-Instruct-2507`	beam_3	0.8000
`Qwen/Qwen3-1.7B`	beam_25	0.5600
InfoBench		
`Qwen/Qwen3-4B`	greedy	0.7882
`Qwen/Qwen3-4B-Instruct-2507`	default	0.7779
`Qwen/Qwen3-1.7B`	beam_3	0.7468

☐

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.

Solution:

Task	Model	Decoding Settings	Score
Graph	`Qwen3-4B`	Self-refine	0.3300
MMLU	`GPT4o-mini`	MCP Deep research	0.8666
InfoBench	`Qwen/Qwen3-4B`	greedy	0.7882

☐

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).

5.3 Planning improvements

Solution:
Graph: Apply speculative decoding with a smaller draft model (Qwen3-0.6B) to rapidly generate candidate path sequences, then verify them using Qwen3-4B combined with a graph-specific verification tool (similar to the MCP-style tool from HW2 Task 2.1) to validate path correctness and weights in parallel. This addresses the current 0.33 accuracy by leveraging the draft models and speculative decoding, enabling faster exploration of the discrete search space while maintaining correctness through structured verification.
MMLU: Implement prefix sharing and KV cache optimization (Nov 11 lecture) to efficiently process multiple reasoning chains with shared system prompts and question prefixes, enabling scaling to 50+ diverse reasoning paths at similar computational cost as current best-of-n (HW2). Combined with training-time distillation techniques to create a smaller student model that maintains the 4B model's medical knowledge while enabling faster inference during the generation phase, this could improve the current 0.87 accuracy through increased sample diversity without proportional cost increases.
InfoBench: Apply diffusion-based language generation to generate complete response drafts through iterative denoising rather than autoregressive token-by-token generation, allowing parallel refinement of the entire response structure and better global coherence. This non-autoregressive approach addresses the length bias issue identified in HW2 analysis (log-probability methods penalizing longer answers) and could improve upon the current 0.79 greedy baseline by enabling bidirectional context modeling for more comprehensive, well-structured answers that InfoBench's rubrics favor. □

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.