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:

□

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 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.

[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.

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:

□

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:

□

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.

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

Solution:

Method	Top-1 score	Avg. rank of best output	Spearman rank correlation
Oracle		1	1.0
`Qwen3-4B` log-probs			
`Qwen3-14B` log-probs			
Scalar reward			
Pairwise reward			
MBR with BLEU			
MBR with BERTScore			

□

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:

□

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:

□

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	
`Qwen3-4B` log-probs	
`Qwen3-14B` log-probs	
Scalar reward	
Pairwise reward	
MBR with BLEU	
MBR with BERTScore	

□

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:

□

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		1	1.0
`Qwen3-4B` log-probs			
`Qwen3-14B` log-probs			
Scalar reward			
Pairwise reward			
MBR with BLEU			
MBR with BERTScore			

n=10:

Method	Top-1 score	Avg. rank of best output	Spearman rank correlation
Oracle		1	1.0
`Qwen3-4B` log-probs			
`Qwen3-14B` log-probs			
Scalar reward			
Pairwise reward			
MBR with BLEU			
MBR with BERTScore			

n=20:

Method	Top-1 score	Avg. rank of best output	Spearman rank correlation
Oracle		1	1.0
`Qwen3-4B` log-probs			
`Qwen3-14B` log-probs			
Scalar reward			
Pairwise reward			
MBR with BLEU			
MBR with BERTScore			

□

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:

□

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:

□

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:

□

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:

□

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:

□

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:

□

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

Variant 2: ...

Variant 3: ...

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

Variant	Acc.	Thinking	Queries	Snippet Titles	Snippet Contents	Final Answers
1	-	-	-	-	-	
2	-	-	-	-	-	
3	-	-	-	-	-	

□

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 pro-

posed 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 :{}
Q 4.3.2. □

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`		
`Qwen/Qwen3-4B-Instruct-2507`		
`Qwen/Qwen3-1.7B`		
MMLU		
`Qwen/Qwen3-4B`		
`Qwen/Qwen3-4B-Instruct-2507`		
`Qwen/Qwen3-1.7B`		
InfoBench		
`Qwen/Qwen3-4B`		
`Qwen/Qwen3-4B-Instruct-2507`		
`Qwen/Qwen3-1.7B`		

□

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			
MMLU			
InfoBench			

□

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:

MMLU:

InfoBench: □

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.