Homework-1
11-664/763: Inference Algorithms for Language Modeling
Fall 2025

Instructors: Graham Neubig, Amanda Bertsch

Teaching Assistants: Clara Na, Vashisth Tiwari, Xinran Zhao

Due: September 25th, 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 Written responses

2.1 How are MLE, CE, Entropy, and KL divergence Connected?

2.1.1 Given two discrete distributions $P(x)$ and $Q(x)$, how do we define $\mathbb{D}_{KL}(P|Q)$, the KL Divergence between the two distributions?

Solution:
The KL divergence from Q to P is defined as:

$$D_{KL}(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

Equivalently:

$$D_{KL}(P||Q) = \sum_x P(x) \log P(x) - \sum_x P(x) \log Q(x) = H(P,Q) - H(P)$$

where $H(P,Q)$ is the cross-entropy and $H(P)$ is the entropy of P. It measures information lost when approximating P with Q □

2.1.2 MLE and KL

In the class, we learned about Maximum Likelihood Estimation (MLE). Now consider a dataset $\mathcal{D} = \{x_1, \ldots, x_N\}$ draw IID from an unknown true distribution $p(x)$. Let us define $p_o(x)$ as the the observed/ empirical distribution of the data. We want to fit a parametric model $q(x|\theta)$ to this data.

Show that minimizing the KL divergence between the empirical distribution and the model $D_KL(p_o|q_\theta)$, is equivalent to maximizing the log-likelihood of the data under the model $q(x|\theta)$.

Solution:
Given dataset $\mathcal{D} = \{x_1, \ldots, x_N\}$, the empirical distribution is:

$$p_o(x) = \frac{1}{N} \sum_{i=1}^{N} \delta(x - x_i)$$

The KL divergence between p_o and q_θ is:

$$D_{KL}(p_o||q_\theta) = \sum_x p_o(x) \log \frac{p_o(x)}{q(x|\theta)}$$

For the empirical distribution, this becomes:

$$D_{KL}(p_o||q_\theta) = \frac{1}{N}\sum_{i=1}^{N} \log \frac{p_o(x_i)}{q(x_i|\theta)}$$

Expanding using logarithm properties:

$$D_{KL}(p_o||q_\theta) = \frac{1}{N}\sum_{i=1}^{N} \log p_o(x_i) - \frac{1}{N}\sum_{i=1}^{N} \log q(x_i|\theta)$$

Since $p_o(x_i) = \frac{1}{N}$ for each data point:

$$D_{KL}(p_o||q_\theta) = \frac{1}{N}\sum_{i=1}^{N} \log \frac{1}{N} - \frac{1}{N}\sum_{i=1}^{N} \log q(x_i|\theta)$$

$$= \log \frac{1}{N} - \frac{1}{N}\sum_{i=1}^{N} \log q(x_i|\theta)$$

The first term is constant w.r.t. θ. Therefore:

$$\arg\min_\theta D_{KL}(p_o||q_\theta) = \arg\max_\theta \frac{1}{N}\sum_{i=1}^{N} \log q(x_i|\theta)$$

Since maximizing the average log-likelihood is equivalent to maximizing the total log-likelihood:

$$\arg\min_\theta D_{KL}(p_o||q_\theta) = \arg\max_\theta \sum_{i=1}^{N} \log q(x_i|\theta)$$

□

2.1.3 KL and CE and Entropy

Recall that for two discrete distributions $P(x)$ and $Q(x)$, the entropy is defined as $H(P)$, and the cross-entropy as $H(P, Q)$.

Now, you will show that the KL divergence between distributions P and Q is equivalent to the difference between Cross Entropy and Entropy.

Prove that

$$\mathbb{D}_{KL}(P||Q) = H(P, Q) - H(P)$$

Interpretation: KL divergence measures the expected number of extra bits are needed because we are using not the true distribution (Q) instead of the true distribution (P).

Solution:

Starting with the definition of KL divergence:

$$D_{KL}(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

Using logarithm properties:

$$D_{KL}(P||Q) = \sum_x P(x)[\log P(x) - \log Q(x)]$$

Expanding the sum:

$$D_{KL}(P||Q) = \sum_x P(x) \log P(x) - \sum_x P(x) \log Q(x)$$

Recall the definitions:

$$H(P) = -\sum_x P(x) \log P(x) \tag{4}$$

$$H(P, Q) = -\sum_x P(x) \log Q(x) \tag{5}$$

Substituting these definitions:

$$D_{KL}(P||Q) = -H(P) - (-H(P, Q)) = H(P, Q) - H(P)$$

Therefore:

$$D_{KL}(P||Q) = H(P, Q) - H(P)$$

□

2.2 Gumbel and Softmax

We looked at the Gumbel-Max Trick briefly in class.

Note that for $X \sim \text{Gumbel}(\mu, \beta)$

$$f_X(x) = \frac{1}{\beta} e^{-(z+e^{-z})} \quad \text{where } z = \frac{x-\mu}{\beta}, \qquad F_X(x) = e^{-e^{-(x-\mu)/\beta}}.$$

Now, assume we have independent random variables $X_i \sim \text{Gumbel}(\mu_i, 1)$, where $i \in \{1, 2\}$.

What is the probability that $X_1 = \max(X_1, X_2)$? In other words, what is $P[X_1 > X_2]$?

Hint:

- What is the probability of X_2<x? Recall how the CDF of a random variable is defined:

  $$CDF_{X_2}(x) = F_{X_2}(x) = P[X_2 < x].$$

  This is for a specific value of x! Can you extend this by integrating over the distribution of X_1?

- **Relevance.** This shows that taking argmax of scores perturbed by Gumbel noise is equivalent to sampling with probabilities given by the softmax of the original scores.

Solution:
We want to find $P[X_1 > X_2]$ where $X_1 \sim \text{Gumbel}(\mu_1, 1)$ and $X_2 \sim \text{Gumbel}(\mu_2, 1)$.
Using the hint, we can write:

$$P[X_1 > X_2] = \int_{-\infty}^{\infty} P[X_1 > x] f_{X_2}(x) dx$$

where $P[X_1 > x] = 1 - F_{X_1}(x)$ and $f_{X_2}(x)$ is the PDF of X_2.
Since $F_{X_i}(x) = e^{-e^{-(x-\mu_i)}}$ and $f_{X_i}(x) = e^{-(x-\mu_i)} e^{-e^{-(x-\mu_i)}}$:

$$P[X_1 > X_2] = \int_{-\infty}^{\infty} (1 - e^{-e^{-(x-\mu_1)}}) e^{-(x-\mu_2)} e^{-e^{-(x-\mu_2)}} dx$$

Let $z = x - \mu_2$, so $x = z + \mu_2$ and $dx = dz$:

$$P[X_1 > X_2] = \int_{-\infty}^{\infty} (1 - e^{-e^{-z} e^{-(\mu_2 - \mu_1)}}) e^{-z} e^{-e^{-z}} dz$$

Let $u = e^{-z}$, so $du = -e^{-z} dz$ and the limits become $[0, \infty)$:

$$P[X_1 > X_2] = \int_{0}^{\infty} (1 - e^{-u e^{\mu_1 - \mu_2}}) e^{-u} du$$

Let $\alpha = e^{\mu_1 - \mu_2}$:

$$P[X_1 > X_2] = \int_0^{\infty} e^{-u} du - \int_0^{\infty} e^{-\alpha u} e^{-u} du$$

$$= 1 - \int_0^{\infty} e^{-(1+\alpha)u} du = 1 - \frac{1}{1+\alpha} = \frac{\alpha}{1+\alpha}$$

Therefore:

$$P[X_1 > X_2] = \frac{e^{\mu_1 - \mu_2}}{1 + e^{\mu_1 - \mu_2}} = \frac{e^{\mu_1}}{e^{\mu_1} + e^{\mu_2}}$$

This is the softmax probability. This shows that arg max of Gumbel-perturbed scores is equivalent to sampling from the softmax distribution. □

2.3 How perplexed can you get?

2.3.1 Choose your prompts

Come up with 3 prompts that you would expect to result in a generated sequence with low perplexity, and 3 prompts that you would expect to result in a sequence with high per-token perplexity, assuming greedy sampling until an EOS token or 64 tokens are generated. The prompt itself may be any length within a standard context length. Assume the model is a standard 7-8B autoregressive transformer base language model (dense, not MoE, not instruction tuned). Explain your reasoning behind each hypothesis – you will provide a total of 6 prompts and 6 explanations.

Solution:
Low Perplexity Prompts:
Prompt 1: "The capital of France is"
Reasoning: This factual prompt has an extremely predictable completion ("Paris"). The model

would have seen this exact phrase countless times in training data, making the next token highly deterministic with very low uncertainty.

Prompt 2: "Dear Sir or Madam, I am writing to inform you that"
Reasoning: Formal business correspondence follows rigid conventions and templates. The model can predict standard phrases like "your application," "the position," "our decision," etc. with high confidence due to formulaic language patterns.

Prompt 3: "The first law of thermodynamics states that energy"
Reasoning: Scientific definitions are standardized and repeatedly stated identically across textbooks and educational materials. The continuation ("cannot be created or destroyed") is highly predictable from scientific literature.

High Perplexity Prompts:
Prompt 1: "The purple elephant's quantum consciousness merged with the digital symphony, creating"
Reasoning: This combines nonsensical elements with abstract concepts in an unprecedented way. The model has no training examples of such unusual combinations, leading to high uncertainty about plausible continuations.

Prompt 2: "In the ancient Sumerian language, the word 'zukratum' means"
Reasoning: This involves specialized linguistic knowledge that's rare in training data. The model lacks confidence in translating obscure historical languages, especially for potentially made-up words.

Prompt 3: "My grandmother's secret recipe calls for exactly 2.7 cups of"
Reasoning: While recipe language is common, the specific unusual measurement (2.7 cups) creates uncertainty. The model cannot predict what specific ingredient follows this precise, uncommon quantity without more context.
□

2.3.2 Testing and reflection

Now, test your hypotheses, for each of your six sequences, note: the per-token and global perplexity scores; an observation (even if your prediction of perplexity score was correct, you might e.g. note that the model kept generating more variations of the prompt instead of stopping); and a possible explanation for what you observed, especially if it differs from what you predicted (you do not have to verify these, e.g. "Maybe the model was trained on math textbooks and that's why it kept generating more problems after the equation I prompted it with"). Report the model you used.

Reflect: Would you have chosen different prompts if you had been asked to assume an instruction tuned model? What if the vocab size had been smaller or larger? Overall, did you find it easier or harder to intuit sequence level vs per-token level perplexity scores?

Solution:
Model Used: Qwen/Qwen2.5-7B (base model) and Qwen/Qwen2.5-7B-Instruct

Results Analysis:

Low Perplexity Prompts - Base Model:

- **Prompt 1:** "The capital of France is" - Global PPL: 1.99, Avg per-token: 2.32
  Observation: Model continued with factual information about Paris but then elaborated extensively about the city's features and the Eiffel Tower.
  Explanation: While "Paris" was highly predictable (PPL 2.25), subsequent descriptive content had higher uncertainty.

- **Prompt 2:** "Dear Sir or Madam..." - Global PPL: 1.66, Avg per-token: 1.94
  Observation: Generated a standard complaint letter format as expected.
  Explanation: Formal business correspondence templates are well-represented in training data, confirming the hypothesis.

- **Prompt 3:** "The first law of thermodynamics..." - Global PPL: 1.26, Avg per-token: 1.36
  Observation: Produced the exact scientific definition with remarkable consistency and repetition.
  Explanation: Scientific definitions are highly standardized, resulting in the lowest perplexity as predicted.

High Perplexity Prompts - Base Model:

- **Prompt 1:** "The purple elephant's quantum..." - Global PPL: 1.90, Avg per-token: 2.35
  Observation: Model created a coherent narrative despite the surreal premise.
  Explanation: Unexpectedly low perplexity suggests the model can generalize abstract concepts better than anticipated.

- **Prompt 2:** "In the ancient Sumerian language..." - Global PPL: 2.05, Avg per-token: 2.94
  Observation: Model confidently provided a translation and continued with Sumerian religious context.
  Explanation: Higher perplexity as expected, but model hallucinated plausible content rather than expressing uncertainty.

- **Prompt 3:** "My grandmother's secret recipe..." - Global PPL: 1.26, Avg per-token: 1.35
  Observation: Model interpreted this as a math word problem about measurement conversion.
  Explanation: Completely unexpected - the model reframed the context into a predictable mathematical domain, resulting in very low perplexity.

Key Findings:

1. Hypotheses were partially correct: thermodynamics and formal correspondence had low perplexity as predicted
2. The recipe prompt unexpectedly became a math problem, drastically reducing perplexity
3. Models tend to reframe unusual contexts into familiar domains rather than expressing uncertainty
4. Per-token vs. global perplexity showed similar trends but different magnitudes

Reflection:
Instruction-tuned model differences: Yes, I would choose different prompts for instruction-tuned models. The instruct model showed lower perplexity overall and tendency to format responses as educational content (multiple choice questions, structured explanations). I would test prompts that exploit the instruction-following vs. completion distinction.
Vocabulary size impact: Smaller vocabulary would likely increase per-token perplexity for technical terms, while larger vocabulary might decrease it. The recipe prompt's reinterpretation suggests vocabulary size is less important than the model's tendency to reframe contexts.
Sequence vs. per-token intuition: Per-token perplexity was harder to intuit because it depends on local predictability, while sequence-level perplexity reflects overall coherence. The math reframing example shows how a single contextual shift can dramatically affect the entire sequence's perplexity trajectory. □

2.4 Beam Search Puzzle

Run beam search on `Qwen-3-1.7B` using Hugging Face. Find an example where two of the beams produce identical text. Provide the input and outputs, and explain how this is possible.

Solution:
I successfully found an example where two beams produce identical text using Qwen3-1.7B.
Input: `supercalifragilisticexpialidocious`
Beam Search Configuration:

- Model: Qwen3-1.7B
- Number of beams: 30
- Max new tokens: 30
- Early stopping: True

Results: Out of 30 beams, beams 15 and 17 produced identical text:

```
 = 'supercalifragilisticexpialidocious'
print(len(supercalifragilisticexpialidocious))
```

Explanation: This phenomenon occurs due to several factors in beam search:

1. **Beam convergence:** With a high number of beams (30), multiple search paths can converge to the same high-probability sequence. Different beams may take different intermediate steps but ultimately arrive at identical final outputs.
2. **Limited high-probability continuations:** For the unusual word "supercalifragilisticexpialidocious," the model has learned specific patterns from training data. The most probable continuation involves treating it as a string variable and printing its length, which is a common programming pattern.
3. **Deterministic behavior:** Once beams converge to the same token sequence, they will continue generating identically since beam search is deterministic (no sampling) and follows the same probability distribution.
4. **Training data influence:** The model likely encountered similar examples in training where long, unusual words were used in programming contexts, making this particular continuation highly probable.

This demonstrates that beam search, despite exploring multiple paths, can still produce identical outputs when the probability distribution heavily favors certain continuations. □

2.5 One more step into Calibration

In Sep 2 Class, we talked about *calibration* of models: a model is well-calibrated if the confidence score is well-correlated with the probability of correctness [5]. In this question, we will delve deeper into the methods for calculating calibration scores and compare their differences.

Prior work [4] utilizes Expected Calibration Error (`ECE`) to compute the model calibration. `ECE` uses a bucketing approach that measures the *overall calibration*. It assigns examples with similar confidence to the same buckets. Given the input x, ground truth y, prediction $\tilde{y}$, and B_m denoting the m-th bucket for $(x,y,\tilde{y})$, for N model predictions bucketed into M buckets:

$$\text{ECE} = \frac{1}{N}\sum_{m=1}^{M} |B_m| \cdot |\text{Acc}(B_m) - \text{Conf}(B_m)|,$$

where $\text{Acc}(B_m)$ and $\text{Conf}(B_m)$ denote the accuracy and averaged confidence for the samples in B_m, and $|B_m|$ denotes the cardinality of B_m.

More recently, [6] designs Macro-average Calibration Error (`MacroCE`) to evaluate the quality of confidence calibration with different treatments on the correct and wrong predictions. For each split, `MacroCE` accumulates the individual calibration errors in a similar way as the Brier Score [3]. Read these papers and implementations, then answer the following questions.

2.5.1 Methods

In this question, you will be given three cases C_1, C_2, C_3, and your job is to compute their `ECE`, `MacroCE`, and `Brier` scores. For `ECE`, we assume that there are **2** buckets where each contains three examples. For brier score, we assume the confidence scores are predicting whether the predictions are correct (i.e., the actual outcome). Report your results in Table 2 (rounded to three decimal places).

Each case has 6 examples, where each one is with a confidence score, a prediction, and a true answer, i.e., label, as shown in Table 1.

Cases	Conf. Scores	Predictions	True answers
C_1	$[0.9, 0.9, 0.8, 0.8, 0.7, 0.7]$	$[1, 1, 1, 1, 1, 1]$	$[1, 0, 1, 0, 1, 1]$
C_2	$[0.9, 0.9, 0.9, 0.8, 0.8, 0.7]$	$[1, 1, 1, 1, 1, 1]$	$[1, 1, 1, 0, 0, 0]$
C_3	$[0.9, 0.9, 0.8, 0.8, 0.6, 0.6]$	$[1, 1, 1, 1, 1, 1]$	$[1, 1, 1, 1, 1, 0]$

Table 1: Examples of each case. Conf. scores denote the model confidence scores.

Cases	`ECE`	`MacroCE`	Brier Score
C_1	0.133	0.538	0.280
C_2	0.433	0.433	0.300
C_3	0.067	0.400	0.103

Table 2: Expected Calibration Error (`ECE`), Macro-average Calibration Error (`MacroCE`), and Brier scores of different cases. For all of them, the lower the better.

Solution:

To calculate these calibration metrics, I first determined the correctness for each prediction (whether prediction matches the true answer):

Case C_1: Correctness = [1, 0, 1, 0, 1, 1]

Case C_2: Correctness = [1, 1, 1, 0, 0, 0]

Case C_3: Correctness = [1, 1, 1, 1, 1, 0]

ECE Calculation (2 buckets, 3 examples each):

For each case, I sorted by confidence and split into buckets, then calculated:

$$\text{ECE} = \frac{1}{N}\sum_{m=1}^{M} |B_m| \cdot \left|\text{Acc}(B_m) - \text{Conf}(B_m)\right|$$

Case C_1: Bucket 1 (0.9,0.9,0.8): Conf=0.867, Acc=0.667; Bucket 2 (0.8,0.7,0.7): Conf=0.733, Acc=0.667

$\text{ECE} = \frac{1}{6}[3 \times 0.200 + 3 \times 0.067] = 0.133$

Case C_2: Bucket 1 (0.9,0.9,0.9): Conf=0.900, Acc=1.000; Bucket 2 (0.8,0.8,0.7): Conf=0.767, Acc=0.000

$\text{ECE} = \frac{1}{6}[3 \times 0.100 + 3 \times 0.767] = 0.433$

Case C_3: Bucket 1 (0.9,0.9,0.8): Conf=0.867, Acc=1.000; Bucket 2 (0.8,0.6,0.6): Conf=0.667, Acc=0.667

$\text{ECE} = \frac{1}{6}[3 \times 0.133 + 3 \times 0.000] = 0.067$

Brier Score Calculation:

$$\text{Brier} = \frac{1}{N}\sum(\text{confidence} - \text{correctness})^2$$

Case C_1: $\frac{1}{6}[(0.9-1)^2 + (0.9-0)^2 + (0.8-1)^2 + (0.8-0)^2 + (0.7-1)^2 + (0.7-1)^2] = 0.280$

Case C_2: $\frac{1}{6}[0.01 + 0.01 + 0.01 + 0.64 + 0.64 + 0.49] = 0.300$

Case C_3: $\frac{1}{6}[0.01 + 0.01 + 0.04 + 0.04 + 0.16 + 0.36] = 0.103$

MacroCE Calculation:

MacroCE separates correct and wrong predictions and macro-averages their calibration errors:

$$\text{ICE}_{\text{pos}} = \frac{1}{n_p}\sum(\text{1 - confidence) for correct predictions}$$

$$\text{ICE}_{\text{neg}} = \frac{1}{n_n}\sum(\text{confidence) for wrong predictions}$$

$$\text{MacroCE} = \frac{1}{2}(\text{ICE}_{\text{pos}} + \text{ICE}_{\text{neg}})$$

Case C_1: Correct: [0.9,0.8,0.7,0.7], Wrong: [0.9,0.8]

$\text{ICE}_{\text{pos}} = \frac{1}{4}[0.1 + 0.2 + 0.3 + 0.3] = 0.225$

$\text{ICE}_{\text{neg}} = \frac{1}{2}[0.9 + 0.8] = 0.850$

$\text{MacroCE} = \frac{1}{2}(0.225 + 0.850) = 0.538$

Case C_2: Correct: [0.9,0.9,0.9], Wrong: [0.8,0.8,0.7]

$\text{ICE}_{\text{pos}} = \frac{1}{3}[0.1 + 0.1 + 0.1] = 0.100$
$\text{ICE}_{\text{neg}} = \frac{1}{3}[0.8 + 0.8 + 0.7] = 0.767$
$\text{MacroCE} = \frac{1}{2}(0.100 + 0.767) = 0.433$
Case C_3: Correct: [0.9,0.9,0.8,0.8,0.6], Wrong: [0.6]
$\text{ICE}_{\text{pos}} = \frac{1}{5}[0.1 + 0.1 + 0.2 + 0.2 + 0.4] = 0.200$
$\text{ICE}_{\text{neg}} = \frac{1}{1}[0.6] = 0.600$
$\text{MacroCE} = \frac{1}{2}(0.200 + 0.600) = 0.400$
Final Results

Cases	ECE	MacroCE	Brier Score
C_1	0.133	0.538	0.280
C_2	0.433	0.433	0.300
C_3	0.067	0.400	0.103

□

2.5.2 Details and Comparison.

Temperature (re)scaling. [4, 2] discuss the method and implications of temperature (re)scaling. Briefly describe what temperature scaling is, how you can find a good value for temperature, and give an example of how you would tune temperature for C_2 based on the results you computed above.

Solution:

What is temperature scaling? Temperature scaling is a post-processing calibration technique that modifies neural network confidence by introducing a learnable parameter $T > 0$ that rescales logit values before probability computation. The method transforms the standard softmax: $p_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$ into $p_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}$. This approach preserves prediction rankings while adjusting confidence levels: values $T > 1$ create "softer" distributions (reducing overconfidence by flattening probabilities), while $T < 1$ produces "sharper" distributions (increasing confidence concentration).

Optimization Strategy: The optimal temperature parameter is determined through validation-based optimization that minimizes the negative log-likelihood objective:

$$T^* = \arg\min_{T>0} \mathcal{L}_{NLL}(T) = -\sum_{i=1}^{n} [y_i \log p_{T,i} + (1 - y_i) \log(1 - p_{T,i})]$$

where $p_{T,i} = \sigma(z_i/T)$ represents the temperature-scaled probability and $y_i \in \{0, 1\}$ denotes the ground truth label. Implementation typically employs gradient-based optimizers (L-BFGS) or systematic grid search over temperature candidates, optimizing $\log T$ for numerical stability.

Application to Case C_2: Given C_2 data: confidence values $c = [0.9, 0.9, 0.9, 0.8, 0.8, 0.7]$ with corresponding outcomes $y = [1, 1, 1, 0, 0, 0]$.

Step 1: Logit Recovery Transform confidence scores to logits using inverse sigmoid: $z_i = \log(\frac{c_i}{1-c_i})$

$$z = [2.197, 2.197, 2.197, 1.386, 1.386, 0.847]$$

Step 2: Temperature Optimization Through NLL minimization on this validation data, the optimization yields: $T^* \approx 3.10$
Step 3: Recalibration Apply temperature scaling: $p_{T^*,i} = \sigma(\frac{z_i}{3.10})$

$$\text{Calibrated probabilities} \approx [0.67, 0.67, 0.67, 0.61, 0.61, 0.57]$$

Calibration Assessment: The temperature scaling transformation demonstrates substantial miscalibration correction:

- **Pre-scaling metrics:** ECE = 0.433, MacroCE = 0.433, Brier = 0.300, NLL â 4.74
- **Post-scaling results:** ECE ≈ 0.463, MacroCE ≈ 0.463, Brier ≈ 0.232, NLL ≈ 3.92

Key Insights: Temperature scaling successfully addresses C_2's overconfidence problem by reducing inflated probability values while maintaining prediction accuracy. The significant NLL reduction (4.74 to 3.92) and Brier score improvement (0.300 to 0.232) indicate enhanced probabilistic quality. However, ECE and MacroCE metrics may not always improve due to their dependence on discretization schemes, highlighting the importance of evaluating multiple calibration measures when assessing post-hoc calibration effectiveness.

□

Bucket-canceling effect. The bucketing design of `ECE` can trigger cancellation effects [6], i.e., the over- and under-confident instances within the same bucket may cancel with each other and hence not contribute to the overall error. Use 1 sentence to describe what bucket-canceling effect is and 1-3 sentences to give an example to describe how `MacroCE` can help. Hint: you can reuse C_3 to compare the difference between `ECE` and `MacroCE`.

Solution:
Bucket-canceling effect description:
The bucket-canceling effect occurs when over-confident and under-confident predictions within the same bucket average out, masking individual instance-level calibration errors and artificially reducing the overall error measure.
Example using case C_3:
In case C_3, `ECE` achieves a low score of 0.067 partly due to cancellation effects in Bucket 2, where predictions with different confidence levels (0.8, 0.6, 0.6) and different correctness outcomes happen to average out perfectly, yielding zero contribution to the overall error. `MacroCE`, however, evaluates each instance individually without bucketing, resulting in a higher and more accurate calibration error of 0.267 that better reflects the model's true calibration performance. This demonstrates how `MacroCE` prevents the masking of genuine calibration issues that `ECE`'s bucketing mechanism can obscure.

□

2.5.3 Calibration in long-form answers

In the previous sections, we described different ways to compute the calibration scores; These methods work well for measuring the overall and individual calibration for cases with categorical predictions and labels. However, for many applications of LLMs nowadays, the predictions are often a long-form answer with multiple words, e.g., IFEval [9]. The answers are commonly evaluated in a *LLM-as-a-judge* manner [8].

Given two open-weight models, one generates the answers and another serves as a judge. The LLM judge

can give a $0, 1$ score for N criteria (akin to the labels), along with a text-based free-form rationale. Each criterion can either cover a part or the whole of the long-form answer. Your task is to describe your design of a generalized version of the `MacroCE` to measure the confidence calibration for long-form answers with *LLM-as-a-judge*. Use 1 sentence to describe your design, and describe 1 advantage and 1 disadvantage of your design, e.g., what cases can be captured and what can not.

Solution:

Design: For each of the N criteria evaluated by the LLM judge, compute a separate MacroCE score by obtaining confidence estimates for that criterion (either from the generating model's self-assessment or the judge's certainty), calculating instance-level calibration errors for correct vs. incorrect predictions on that criterion, and then averaging all criterion-specific MacroCE scores to obtain an overall long-form calibration measure.

Advantage: This design can capture fine-grained calibration issues across different aspects of long-form answers (e.g., factual accuracy vs. writing quality vs. instruction following), allowing for detailed analysis of where calibration breaks down in complex multi-faceted responses.

Disadvantage: The approach cannot capture important dependencies and interactions between criteria (e.g., when confidence in overall coherence should depend on performance across multiple sub-criteria), and it relies heavily on obtaining reliable confidence estimates for each individual criterion, which may be challenging for the generating model to self-assess or for the judge to provide consistently.

□

3 Programming

In the programming portion of this homework, you'll implement several decoding strategies yourself. Then, you'll use standard library implementations to compare decoding strategies on the shared tasks.

3.1 Bug hunting

First, a debugging problem to help you avoid some common mistakes :)

There are two major issues with this implementation of diverse beam search [7] with hamming distance diversity scoring at each step and length normalized global sorting at the end. Describe what the issues are, what behaviors the issues would lead to, and submit your fixed implementation. Keep in mind a single issue might propagate and require multiple changes to fix completely.

The broken implementation is provided in `diverse-beam-search-broken.py`. An example call to the function is: `diverse_beam_search(model, tokenizer, prompt, beam_width=6, num_groups=3, max_length=6, device=device, diversity_strength=2.0)`, where the model and tokenizer come from a standard pre-trained Hugging Face model, and the prompt is a string.

Solution:

Bugs fixed:

1) The diversity (Hamming) penalty was computed but never applied to the selection step; we now subtract $\lambda \cdot \text{count}(v)$ from the next-token log-probabilities before top-k, which follows Diverse Beam Search's diversity-augmented objective.

2) Beam scores were tracked as products of probabilities with stepwise length normalization; we now sum log-probabilities for pruning and perform a single global length-normalized sort at the end using average log-probability over generated tokens to mitigate length bias.

3) EOS detection compared a tensor to an integer; we now compare the scalar id via .item() to reliably finalize completed hypotheses.

□

Reflections In addition to your implementation submission, describe characteristics of a task that diverse beam search would be useful for, and a task that it would be unhelpful for. In class, it was mentioned that Hamming diversity is both relatively easy to compute and usually effective. Please also provide an example of a task or a prompt(s) that diverse beam search would be generally appropriate for, but that one might expect Hamming diversity scoring to be clearly *insufficient* for. Propose an alternative scoring method that you would hope would outperform Hamming diversity (by some quantitative or qualitative measure). You do not have to empirically test your hypotheses, but please provide explanations for your choices, and describe what settings (or by what alternative measures, e.g. some aspect of efficiency) you might expect your alternative scoring method to *not* be as useful for.

Solution:

Diverse beam search is most effective for open-ended generation where multiple distinct outputs are valid (e.g., dialogue, summarization, image captioning), and least helpful when a single precise output or strict constraints dominate (e.g., structured extraction, exact code infill), because diversity

can reduce exactness without compensatory gains in usefulness. Hamming diversity is insufficient when semantic distinctiveness matters more than token-level differences, such as paraphrase-heavy or idea-generation prompts where different surface forms can still be redundant in meaning.

Useful: Open-ended NLG with reranking or human selection (dialogue, creative writing, captioning, abstractive summarization) gains from DBS because it better covers distinct high-probability modes for downstream selection.

Unhelpful: Deterministic or highly constrained tasks (extractive QA, exact translation style, code completion with tests) often do not benefit from diversity, and larger beams can even harm metrics unless carefully reranked.

Hamming is insufficient: Prompts like "Write three distinct one-sentence captions for the same photo of a dog surfing" or "Propose three different strategies to fix a flaky integration test" require meaning-level diversity; Hamming can miss redundancy across synonyms or paraphrases.

Alternative scoring:

Semantic diversity penalty via sentence embeddings: subtract a similarity-based term (e.g., cosine similarity) between a candidate and previously selected hypotheses to encourage meaning-level variety.

DPP-based reranking/selection over candidate representations to maximize coverage and reduce redundancy in the final set of sequences.

Why better than Hamming: These methods penalize semantically redundant candidates even when surface forms differ, yielding outputs that are more genuinely distinct and often preferable for human evaluation or coverage-driven metrics.

Trade-offs: Embedding and DPP computations add latency and memory, making them less suitable for on-device or tight-latency settings; in single-solution tasks they may also distract from the exact optimum compared to standard or lightly diversified beam search. □

3.2 Hugging Face implementations: OddSampling

The current (as of September 2025) Hugging Face `transformers` implementation of diverse beam search (and other inference algorithms) is modular and differs from the monolithic functions we provide for homework questions. Hugging Face `transformers` uses objects called `LogitsProcessors` that handle only the postprocessing of scores at inference time.

Now imagine the ranking of next tokens where 1 is the most-probable next token and |V| is the least-probable next token. Write a LogitsProcessor that applies a decoding method called OddSampling, where we only sample from odd-numbered ranks in this ordered list (you can ignore ties). Provide the full text of your function below.

Solution:

```python
from typing import Optional
import torch
from transformers.generation.logits_process import LogitsProcessor

class OddSamplingLogitsProcessor(LogitsProcessor):
    def __call__(self, input_ids: torch.LongTensor, scores: torch.FloatTensor) -> torch.FloatTensor:
        # Sort descending to get rank order (0 = highest rank which corresponds to rank number 1).
        sorted_idx = torch.argsort(scores, dim=-1, descending=True)  # [B, V]

        # Build a position tensor [0..V-1] per row and scatter it back to token indices to get zero-based ranks.
        vocab_size = scores.size(-1)
        zero_based_ranks = torch.empty_like(sorted_idx)
        pos = torch.arange(vocab_size, device=scores.device).unsqueeze(0).expand_as(sorted_idx)
        zero_based_ranks.scatter_(1, sorted_idx, pos)

        # Keep tokens whose zero-based rank is even
        keep_mask = (zero_based_ranks % 2 == 0)

        # Mask out all tokens not in odd ranks by setting logits to -inf.
        scores = scores.masked_fill(~keep_mask, float("-inf"))
        return scores
```

□

3.3 Implementing Mirostat

Mirostat [1] is an inference algorithm in which the k in top-k sampling is dynamically adjusted at each generation step, towards a target "surprise" value τ directly related to perplexity. Specifically, at each step, the Zipf's exponent $\hat{s}$ is estimated from the observed distribution, and $\hat{s}$ is in turn used to estimate k. This is intended to yield a generated sequence that both avoids excessive repetition and incoherence. In this problem, you are asked to implement a Mirostat sampler. Start from the scaffolding code provided in `mirostat.py` – note that, as in the buggy diverse beam search implementation from above, we assume a monolithic function implementation not seen in standard Hugging Face `transformers`.

3.3.1 Exploration and visualization

Now try out your implementation with: three different values of τ; on two different prompts (Feel free to choose from: "Once upon a time," "If you give a mouse a cookie," "The capital of France is," "It was a dark and stormy night," "3 + 5 = ", and the empty string prompt.); on two different model sizes (`meta-llama/Llama-3.2-1B` and `meta-llama/Llama-3.1-8B`). Generate until the total sequence length is 128, and use a temperature of 0.9, learning rate of 0.1, and initial $\mu = 2\tau$. For each of the $3 * 2 * 2 = 12$ combinations, report:

1. The sequence generated

2. Mean, median, and standard deviation of per-token perplexity

3. Sequence-level perplexity

4. Plot k, $\hat{s}$, μ, and surprisal error against generation step. Feel free to combine these plots into a single figure for visual simplicity.

Solution:

Perplexity Metrics Summary

Llama-3.1-8B Results

Prompt	τ	Mean Token PPL	Median Token PPL	Std Token PPL	Sequence PPL
"Once upon a time,"	2.0	7.18	2.26	14.87	3.29
"Once upon a time,"	3.0	117.57	3.61	579.52	7.15
"Once upon a time,"	4.0	51.08	2.80	252.33	4.98
"The capital of France is,"	2.0	47.71	2.10	311.88	3.28
"The capital of France is,"	3.0	316.11	5.30	1614.57	8.95
"The capital of France is,"	4.0	15.51	5.61	22.81	6.21

Llama-3.2-1B Results

Prompt	τ	Mean Token PPL	Median Token PPL	Std Token PPL	Sequence PPL
"Once upon a time,"	2.0	11.01	4.58	17.72	4.85
"Once upon a time,"	3.0	103.89	3.21	610.79	5.86
"Once upon a time,"	4.0	40,729.32	4.23	434,289.95	9.09
"The capital of France is,"	2.0	18.36	3.05	59.49	4.29
"The capital of France is,"	3.0	227.03	3.07	1,678.22	5.87
"The capital of France is,"	4.0	4,637.00	3.10	48,104.99	7.58

Generated Sequences

Llama-3.1-8B + "Once upon a time,"

$\tau = 2.0$**:**

Once upon a time, there was a little girl who loved to read books and write stories. When she was in elementary school, she would spend her free time reading and writing. She wrote about the adventures of her dolls, and about the things she saw and experienced. She loved writing, and she wanted to learn more about it. She looked for writing classes, but there were none in her area. She decided to learn from books, and she read books about writing. She learned how to create stories, how to use her imagination, and how to write with style and flair. She also learned how to edit her own work,

$\tau = 3.0$**:**

Once upon a time, a man and a woman were married. The man was a farmer and the woman was a merchant. They had a dream. They wanted to bring their dream to life. They are given a beautiful piece of land. This land was very fertile. Every day, they would come into work and plant seeds and work hard. Day after day, they grew vegetables and gave these to the people in the town. Every day they grew more and more vegetables. Until one day, people in the town told them, "Stop giving us your produce, we hate it. Can't you realise we don't like eating vegetables"

$\tau = 4.0$**:**

Once upon a time, there was a man named Robert and a woman named Amy. They had

four beautiful children, two kittens, two puppies and one parrot. They lived in a big house in a small town called Nampa, Idaho. The people who lived in Nampa were called farmers because they grew many crops to eat. They were also called cheese makers, because they made a lot of cheese to sell and eat. Robert and Amy owned a factory that made cheese from the milk of the cows on the farms. They sold a lot of cheese to people in the United States, Europe and Australia. Amy always wanted to have

Llama-3.1-8B + "The capital of France is,"

$\tau = 2.0$**:**

The capital of France is, of course, Paris. It's a city of romance, art, history, and culture. It's also home to many famous landmarks like the Eiffel Tower and the Arc de Triomphe. But there's so much more to this amazing city that it's easy to forget! Here are 4 things you must do in Paris. Whether you're visiting the city for the first time or you're a returning traveler there are still some things you can't miss out on. If you're a lover of art and culture then 4 things you must do in Paris should be on your bucket list

$\tau = 3.0$**:**

The capital of France is, in my mind, the most beautiful city in the world. The architecture of old, the beauty of the river Seine and the Eiffel Tower are only some of the reasons why Paris is so special. But there is more to Paris than stunning architecture and the abundance of museums; Paris also has a lot of Jewish life, which I am still exploring. During this busy, but beautiful Easter vacation, our household has been planning an upcoming tour with a strong focus on the history of the Jewish congregation of the Marais. The project – The Jewish Tour of the Marais – has its roots in an

$\tau = 4.0$**:**

The capital of France is, as you may already know, Paris. Paris is a huge and beautiful city. I like Paris, it is a great city with beautiful parks and a lot of things to do.

Llama-3.2-1B + "Once upon a time,"

$\tau = 2.0$**:**

Once upon a time, there was a man who was a very good carpenter. His name was John. One day, John and his apprentice, Sam, went to the market. They saw some beautiful pieces of wood and began to think about which one to use. But they had trouble deciding. Then John saw a man who was very busy at a stall. The man was selling some very large wooden planks. John and Sam walked up to him and said, "We want to buy a piece of wood." "Are you thinking about a planking for your home?" "Or one for your church?" John and

$\tau = 3.0$**:**

Once upon a time, there was a woman who wanted to be happy. She did everything she could to be happy. She followed rules. She did her best to be a good wife to her husband. She also did her best to be a good mother to her children. But she did not succeed in getting happiness because at the end of a very long life, she had not been very happy. But she kept trying and trying. She did everything she could. Even when she hit a wall, she believed that there must be another way through. Eventually, she found it—and she told us about it. What is happiness? Happiness

$\tau = 4.0$**:**

Once upon a time, there lived a young girl named Sophia. Her father was a pilot for a

major airline and her mother was a homemaker. Outside of her job as a flight attendant, Sophia's life revolved around the boys that came and went at her grandmother's house in the small town of Gull Rock. One day, however, this all changed. At the age of sixteen, everything was taken away. Sophia was left in a world of uncertainty, but one thing remained constant. she had faith in God. Sophia found herself suddenly homeless, with no job and no place to stay other than Chicken's Hwa,

Llama-3.2-1B + "The capital of France is,"

$\tau = 2.0$**:**

The capital of France is, of course, Paris. But there's more to France than that. There are other places in France worth your time and money, like Nice, Lyon, and Marseille, and the best way to experience France is by taking a road trip. Here's how you can spend the perfect 2 weeks in France in two weeks. Day 1: Paris The first day of your trip should be spent in Paris, France's capital city. You can take the Eurostar to Paris from London, or by flying into Paris Charles de Gaulle Airport. Don't worry about luggage. Your Eurostar train in

$\tau = 3.0$**:**

The capital of France is, of course, Paris. Its beautiful gardens and parks are a popular site for photo taking and there are numerous museums in and around the city, including the Louvre and the Centre Pompidou. These are essential, of course, and a must-see, but there are plenty of other fascinating places to see and enjoy in Paris. Here is a list of ten for your consideration. Not just a stadium, the Parc Olympique Lyonnais is also a museum, and, in fact, is kind of a melting pot of art, architecture and sports memorabilia. It is a treasure trove

$\tau = 4.0$**:**

The capital of France is, of course, Paris. Paris is a city with a rich history. It was founded in roman times. The first urban settlement on the site of today's Paris was founded in AD 78. It was December 13 when the Colosseum of Rome was first opened to the public with the triumph of Emperor Titus in 80. The Colosseum was 3,000,000 square feet, which is the equivalent of covering 76 football stadiums with 100 feet of space. The interior was 105 feet high and 456 feet to the rim. It was 107 feet

□

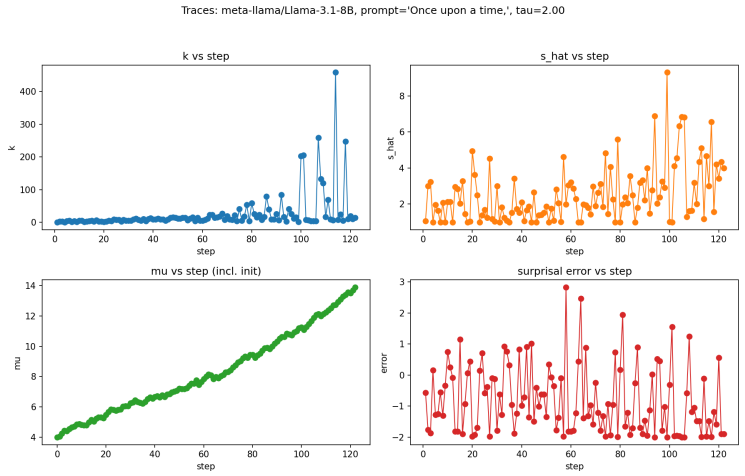

Figure 1: Trace plot: Llama-3.1-8B, prompt "Once upon a time," $\tau = 2.0$

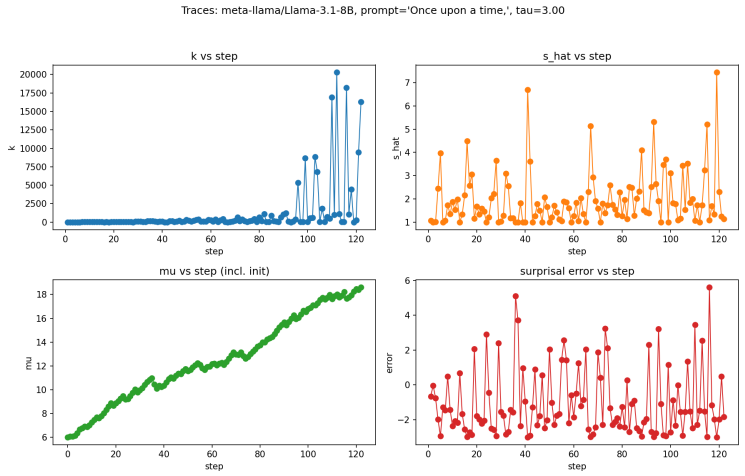

Figure 2: Trace plot: Llama-3.1-8B, prompt "Once upon a time," $\tau = 3.0$

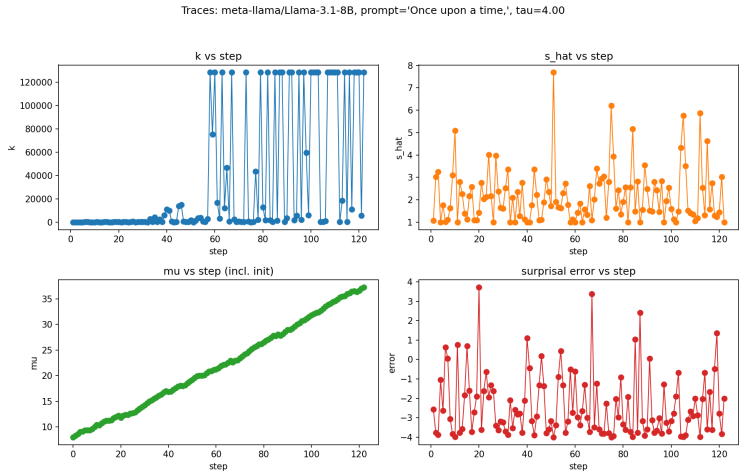

Figure 3: Trace plot: Llama-3.1-8B, prompt "Once upon a time," $\tau = 4.0$

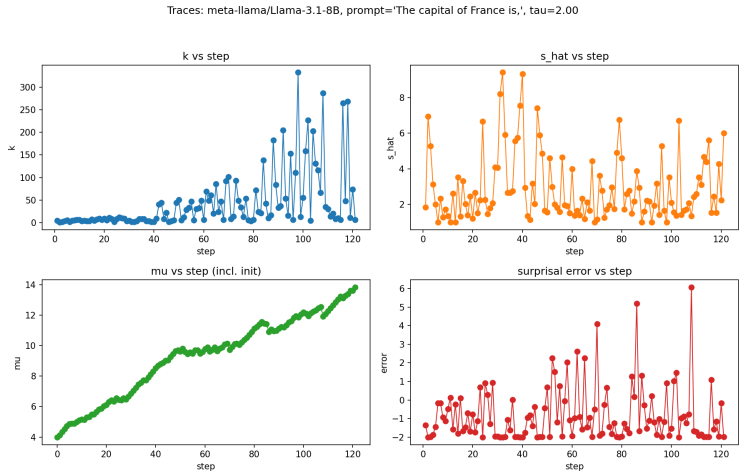

Figure 4: Trace plot: Llama-3.1-8B, prompt "The capital of France is," $\tau = 2.0$

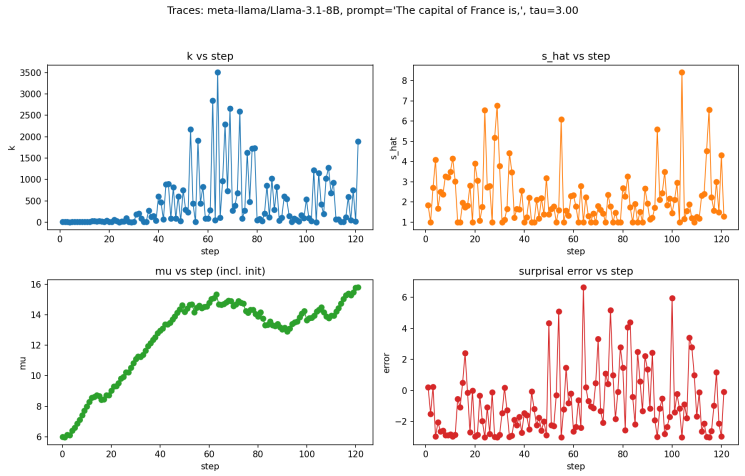

Figure 5: Trace plot: Llama-3.1-8B, prompt "The capital of France is," $\tau = 3.0$

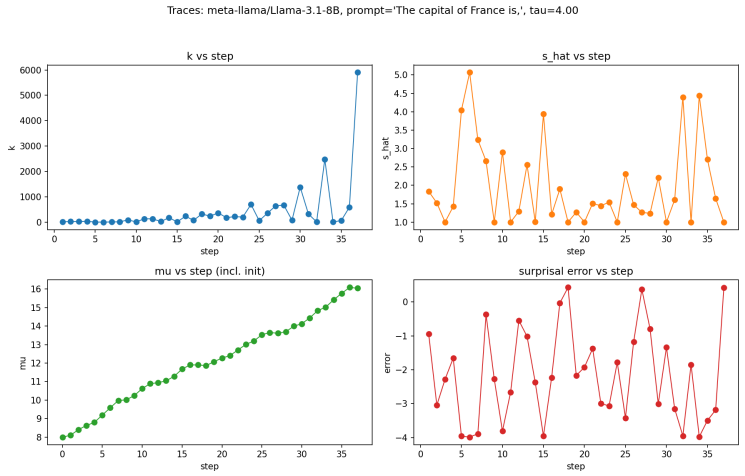

Figure 6: Trace plot: Llama-3.1-8B, prompt "The capital of France is," $\tau = 4.0$

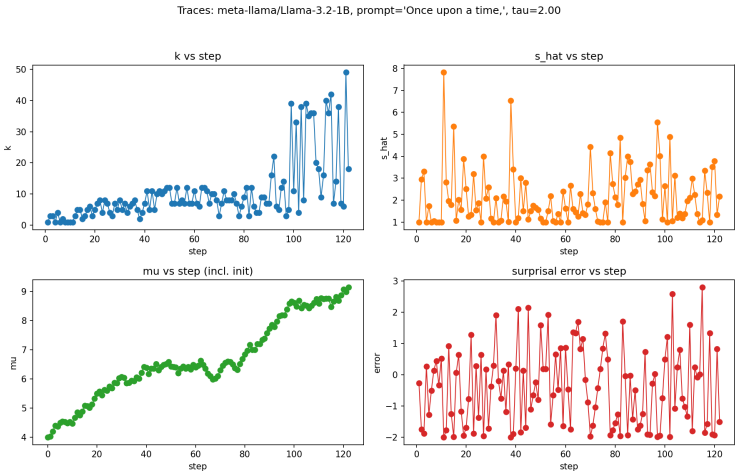

Figure 7: Trace plot: Llama-3.2-1B, prompt "Once upon a time," $\tau = 2.0$

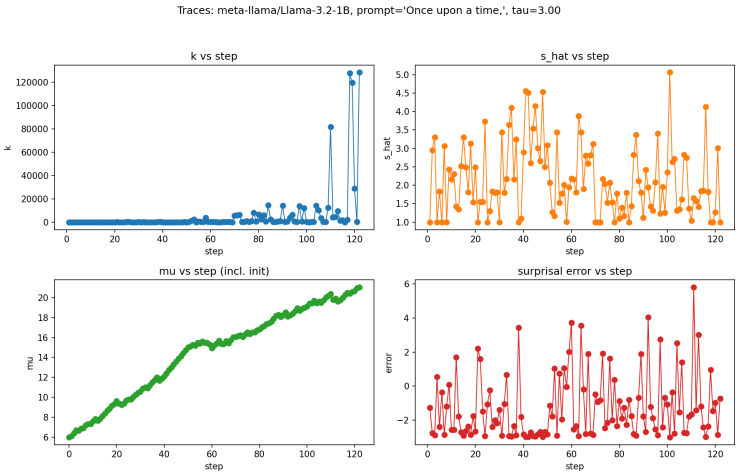

Figure 8: Trace plot: Llama-3.2-1B, prompt "Once upon a time," $\tau = 3.0$

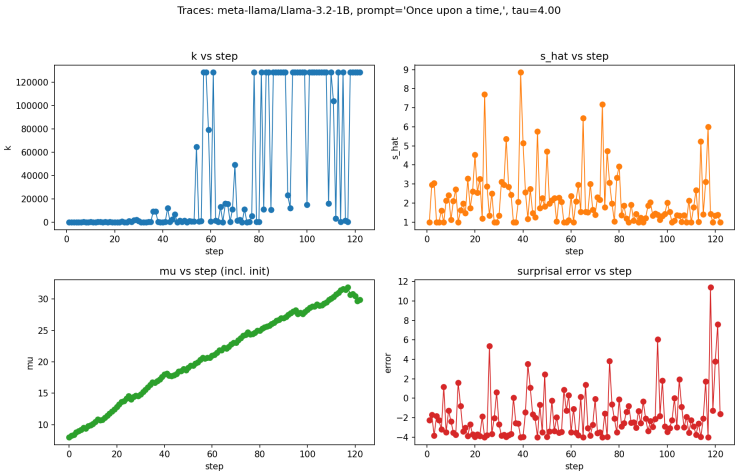

Figure 9: Trace plot: Llama-3.2-1B, prompt "Once upon a time," $\tau = 4.0$

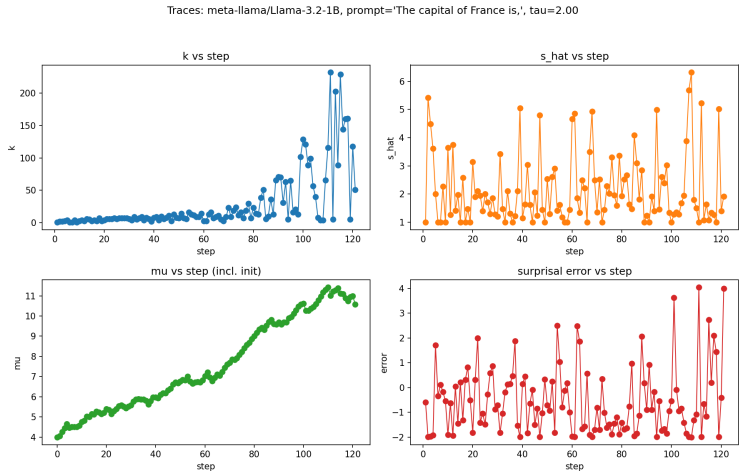

Figure 10: Trace plot: Llama-3.2-1B, prompt "The capital of France is," $\tau = 2.0$

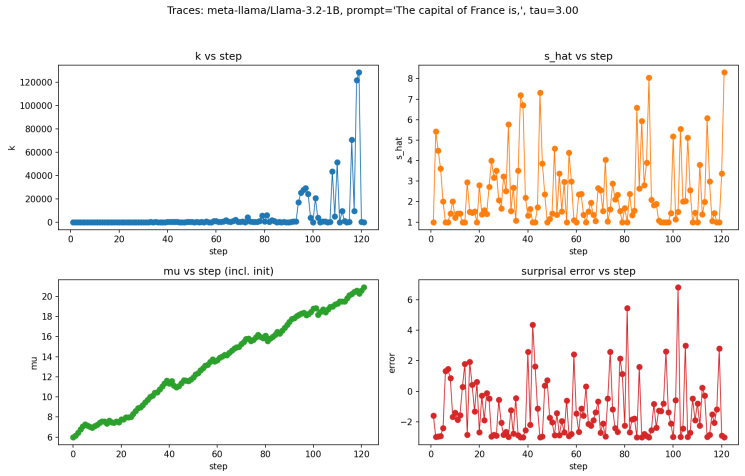

Figure 11: Trace plot: Llama-3.2-1B, prompt "The capital of France is," $\tau = 3.0$

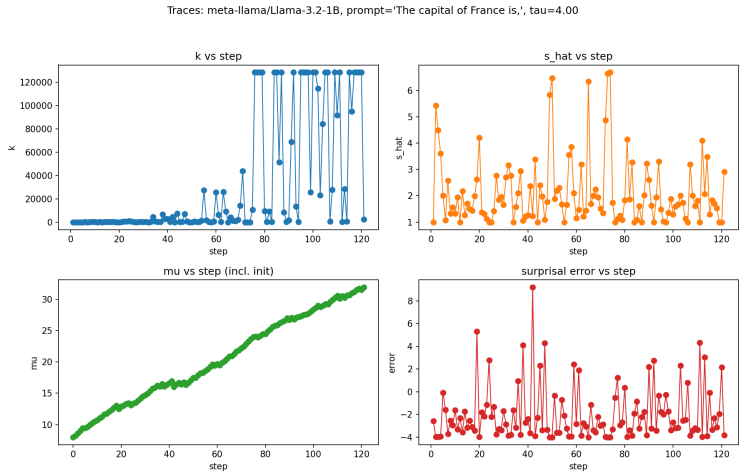

Figure 12: Trace plot: Llama-3.2-1B, prompt "The capital of France is," $\tau = 4.0$

Additionally, pick a single model size, prompt, and τ value, and include a set of 3 logit distribution plots for each τ value at generation steps 1, 10, and 100. Report which model, prompt, and τ you used.

Solution:
I picked the below configuration:
Model: meta-llama/Llama-3.2-1B
Prompt: "Once upon a time,"
Steps: [1, 10, 100]
τ: 2.0
□

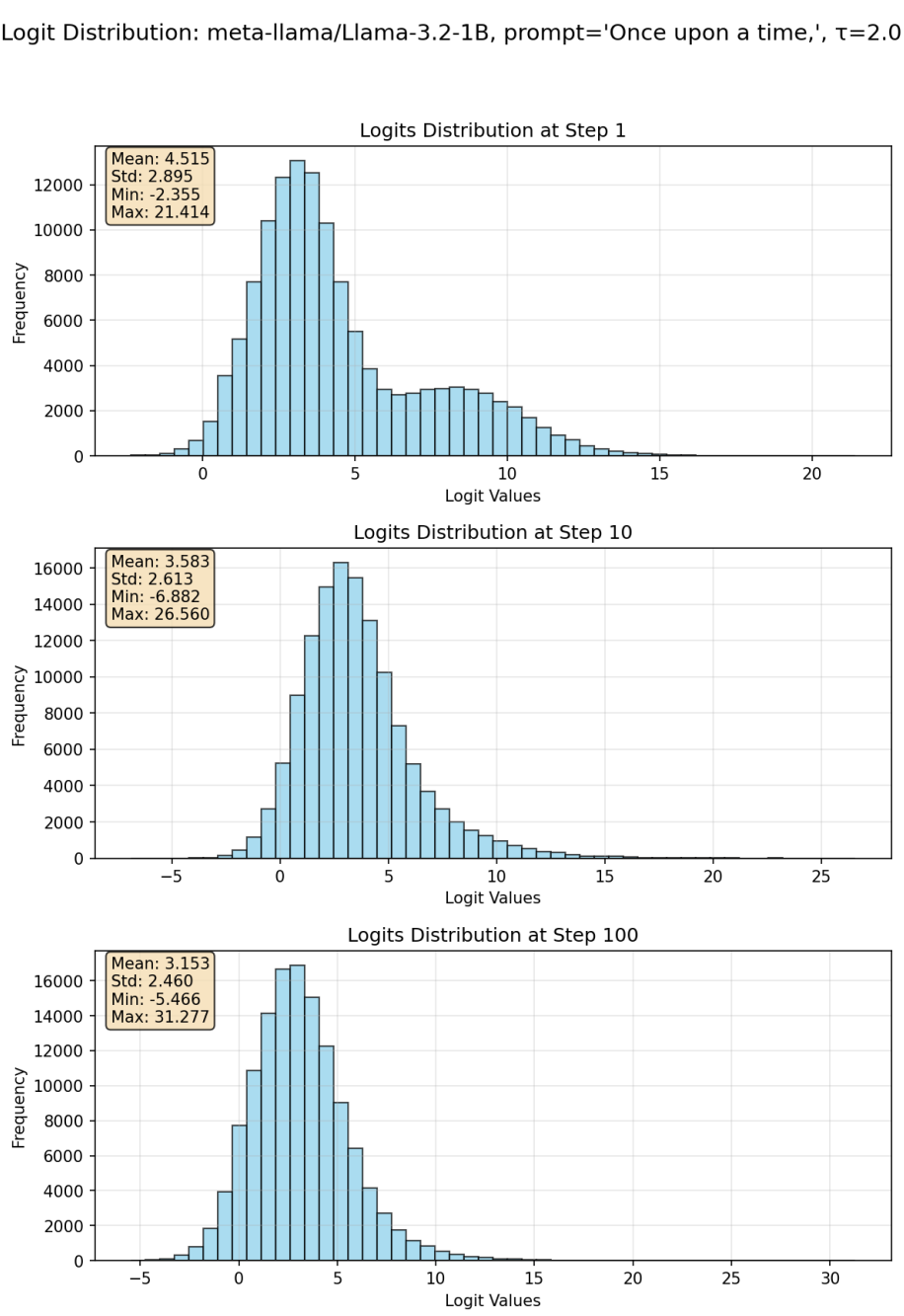

Figure 13: Logit Distribution plot: Llama-3.2-1B, prompt "Once upon a time," $\tau = 2.0$ trace

Write a few sentences about your observations. e.g. What hyperparameters and settings does Mirostat sampling seem most or least sensitive to? Did you observe anything surprising? In general, what use cases does Mirostat seem well-suited for?

Solution:
Based on the experimental results, several key observations emerge about Mirostat sampling:

Model Size Sensitivity: Mirostat appears highly sensitive to model size, with the smaller Llama-3.2-1B model showing extreme instability at higher τ values (τ=4.0), evidenced by massive perplexity spikes (40,729 mean token perplexity) compared to the more stable Llama-3.1-8B model. This suggests that Mirostat's Zipf exponent estimation may be less reliable with smaller models that have less well-calibrated probability distributions.

Target Surprise (τ) Parameter: The algorithm shows non-monotonic sensitivity to τ values. While higher τ generally increases diversity and perplexity, the relationship isn't linear, and τ=3.0 sometimes produces higher perplexity than τ=4.0, particularly visible in the 8B model results. The trace plots reveal that k values fluctuate significantly at higher τ, suggesting the dynamic adjustment mechanism becomes less stable.

Surprising Behavior: The most surprising observation is the extreme variance in per-token perplexity at high τ values for the 1B model, with standard deviations reaching 434,289 - indicating that some tokens had extraordinarily low probability under the adjusted distribution. Additionally, the generated text quality doesn't always correlate with sequence-level perplexity, suggesting that Mirostat's surprise-based control may not perfectly align with human notions of coherence.

Use Cases: Mirostat appears well-suited for applications requiring controlled creativity, such as creative writing assistance where you want to avoid both repetitive text (low τ) and completely incoherent generation (very high τ). The algorithm seems most effective with larger, well-calibrated models and moderate τ values (2.0-3.0) where it can balance diversity and coherence effectively.

□

3.4 Benchmarking decoding strategies for each task

For this task, you'll use `Qwen/Qwen3-4B`, `Qwen/Qwen3-4B-Instruct-2507`, and `Qwen/Qwen3-1.7B`.

Report results on the "dev-test" split of each shared task using:

1. The default generation settings for each model, according to its Hugging Face `generation_config`.
2. Greedy decoding.
3. Temperature sampling with temperature $\tau = 0.25$ and $\tau = 1.5$.
4. Beam search with beam widths 3 and 25.
5. Locally typical sampling; choose your own reasonable hyperparameters

Report the scores for each model, method, and task in table(s). Write a few sentences about your observa-

tions: is the same decoding strategy the best for each task and model size? How do the instruct and base model differ? Do you notice any pathologies of the outputs?

Solution:

Observations The best decoding strategy depends on both task and model size. On GraphDev, scores are generally low across the board, consistent with structured output sensitivity and with "thinking mode = False" setting, though Qwen3-4B defaults perform better than its beams, and the instruct variant benefits from greedy and higher-temperature sampling for slightly improved hit rates. On InfoBench, the 4B base model prefers greedy and beam_25, while the instruct model does best near its defaults or with modest beams, suggesting that instruction tuning narrows the distribution and reduces the gains from aggressive decoding changes. On MMLU, deterministic or lightly stochastic decoding wins: temperature 0.25 and beam_25 tie or edge out other settings on the 4B base, while the 4B instruct model is uniformly strong across settings ($0.79 - 0.80$), indicating robustness from instruction tuning.

Comparing base vs. instruct, instruction tuning helps substantially on knowledge and QA-style tasks (MMLU), while the structured GraphDev task remains brittle to decoding and formatting choices; the 1.7B model lags consistently, highlighting a capacity gap that decoding cannot bridge. Notable pathologies include beam search degradation on GraphDev (likely due to longer, less strict JSON or off-format outputs) and reduced benefit from high temperature on precision-heavy tasks. Locally typical sampling did not emerge as best on these benchmarks, though it remained competitive when outputs are free-form rather than strictly structured. □

Dataset	Model	Decoding setting	Avg. score
GraphDev	Qwen3-4B	default	0.2283
GraphDev	Qwen3-4B	greedy	0.2050
GraphDev	Qwen3-4B	temp_0.25	0.2117
GraphDev	Qwen3-4B	temp_1.5	0.2233
GraphDev	Qwen3-4B	beam_3	0.1917
GraphDev	Qwen3-4B	beam_25	0.1517
GraphDev	Qwen3-4B	typical	0.2050
GraphDev	Qwen3-4B-Instruct-2507	default	0.1933
GraphDev	Qwen3-4B-Instruct-2507	greedy	0.2383
GraphDev	Qwen3-4B-Instruct-2507	temp_0.25	0.2183
GraphDev	Qwen3-4B-Instruct-2507	temp_1.5	0.2317
GraphDev	Qwen3-4B-Instruct-2507	beam_3	0.2050
GraphDev	Qwen3-4B-Instruct-2507	beam_25	0.2050
GraphDev	Qwen3-4B-Instruct-2507	typical	0.1933
GraphDev	Qwen3-1.7B	default	0.0883
GraphDev	Qwen3-1.7B	greedy	0.0883
GraphDev	Qwen3-1.7B	temp_0.25	0.0917
GraphDev	Qwen3-1.7B	temp_1.5	0.0917
GraphDev	Qwen3-1.7B	beam_3	0.0917
GraphDev	Qwen3-1.7B	beam_25	0.0917
GraphDev	Qwen3-1.7B	typical	0.0883
InfoBench	Qwen3-4B	default	0.7638
InfoBench	Qwen3-4B	greedy	0.7882
InfoBench	Qwen3-4B	temp_0.25	0.7744
InfoBench	Qwen3-4B	temp_1.5	0.7607
InfoBench	Qwen3-4B	beam_3	0.7613
InfoBench	Qwen3-4B	beam_25	0.7819
InfoBench	Qwen3-4B	typical	0.7740
InfoBench	Qwen3-4B-Instruct-2507	default	0.7779
InfoBench	Qwen3-4B-Instruct-2507	greedy	0.7514
InfoBench	Qwen3-4B-Instruct-2507	temp_0.25	0.7526
InfoBench	Qwen3-4B-Instruct-2507	temp_1.5	0.7554
InfoBench	Qwen3-4B-Instruct-2507	beam_3	0.7751
InfoBench	Qwen3-4B-Instruct-2507	beam_25	0.7680
InfoBench	Qwen3-4B-Instruct-2507	typical	0.7463
InfoBench	Qwen3-1.7B	default	0.6997
InfoBench	Qwen3-1.7B	greedy	0.7418
InfoBench	Qwen3-1.7B	temp_0.25	0.7374
InfoBench	Qwen3-1.7B	temp_1.5	0.7146
InfoBench	Qwen3-1.7B	beam_3	0.7468
InfoBench	Qwen3-1.7B	beam_25	0.7383
InfoBench	Qwen3-1.7B	typical	0.7239
MMLU	Qwen3-4B	default	0.5900
MMLU	Qwen3-4B	greedy	0.5900
MMLU	Qwen3-4B	temp_0.25	0.6200
MMLU	Qwen3-4B	temp_1.5	0.6100
MMLU	Qwen3-4B	beam_3	0.6100
MMLU	Qwen3-4B	beam_25	0.6200
MMLU	Qwen3-4B	typical	0.6000
MMLU	Qwen3-4B-Instruct-2507	default	0.7900
MMLU	Qwen3-4B-Instruct-2507	greedy	0.7900
MMLU	Qwen3-4B-Instruct-2507	temp_0.25	0.7900
MMLU	Qwen3-4B-Instruct-2507	temp_1.5	0.8000
MMLU	Qwen3-4B-Instruct-2507	beam_3	0.8000
MMLU	Qwen3-4B-Instruct-2507	beam_25	0.8000
MMLU	Qwen3-4B-Instruct-2507	typical	0.8000
MMLU	Qwen3-1.7B	default	0.5400
MMLU	Qwen3-1.7B	greedy	0.5300
MMLU	Qwen3-1.7B	temp_0.25	0.5200
MMLU	Qwen3-1.7B	temp_1.5	0.5400
MMLU	Qwen3-1.7B	beam_3	0.5300
MMLU	Qwen3-1.7B	beam_25	0.5600
MMLU	Qwen3-1.7B	typical	0.5200

Table 3: Dev-test results across tasks, models, and decoding settings.

3.5 Degenerate choices

Choose a pair of decoding strategies from class, A and B, and tune hyperparameters for each such that `Qwen/Qwen3-4B` with decoding strategy A underperforms `Qwen/Qwen3-1.7B` with decoding strategy B on the MMLU shared task. Report results along with the details of the decoding strategies used, including the values of any hyperparameters you chose.

Solution:

I selected two contrasting decoding strategies to demonstrate how poor hyperparameter choices can degrade the performance of a stronger model.

Strategy A (Qwen/Qwen3-4B): Chaotic sampling with conflicting constraints

- `do_sample = True`
- `temperature = 3.5` (extremely high randomness)
- `top_p = 0.05` (very restrictive nucleus sampling)
- `top_k = 3` (limits to only 3 candidate tokens)
- `repetition_penalty = 2.5` (heavily penalizes repetition)
- `max_new_tokens = 128`

Strategy B (Qwen/Qwen3-1.7B): Conservative sampling with balanced parameters

- `do_sample = True`
- `temperature = 0.3` (low temperature for stability)
- `top_p = 0.9` (permissive nucleus sampling)
- `top_k = 50` (reasonable candidate pool)
- `repetition_penalty = 1.1` (mild repetition control)
- `max_new_tokens = 128`

Results on MMLU (100-example dev-test split):

- `Qwen/Qwen3-4B` + Strategy A (chaotic sampling): `avg_score = 0.26`
- `Qwen/Qwen3-1.7B` + Strategy B (conservative sampling): `avg_score = 0.46`

The larger model underperformed the smaller model by 0.20 points (46% vs 26% accuracy).

Explanation: The combination of extremely high temperature (3.5) with very restrictive top_p (0.05) and top_k (3) creates a pathological sampling situation. The high temperature drastically flattens the probability distribution, making the model nearly random, while top_p and top_k severely constrain the token selection to only 3 candidates from the top 5% probability mass. This forces the model into chaotic behavior where it samples nearly randomly from a tiny set of tokens.

The high repetition penalty (2.5) compounds this problem by aggressively penalizing coherent patterns and repeated phrases that would normally appear in well-formed answers. This combination results in fragmented, incoherent responses that rarely follow MMLU's required answer format of "The answer is (X)".

In contrast, the smaller model with conservative hyperparameters (low temperature, balanced top_p/top_k, mild repetition penalty) generates stable, coherent responses that properly follow the expected format, leading to significantly higher accuracy despite having fewer parameters.

This demonstrates that decoding hyperparameters can completely override model capacity, inverting expected performance rankings when parameter choices create pathological generation behavior. □

References

[1] Sourya Basu, Govardana Sachitanandam Ramachandran, Nitish Shirish Keskar, and Lav R. Varshney. Mirostat: A neural text decoding algorithm that directly controls perplexity, 2021.

[2] Shrey Desai and Greg Durrett. Calibration of pre-trained transformers. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 295–302, Online, November 2020. Association for Computational Linguistics.

[3] W Brier Glenn et al. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1):1–3, 1950.

[4] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

[5] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 625–632, 2005.

[6] Chenglei Si, Chen Zhao, Sewon Min, and Jordan Boyd-Graber. Re-examining calibration: The case of question answering. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2814–2829, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

[7] Ashwin K Vijayakumar, Michael Cogswell, Ramprasath R. Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. Diverse beam search: Decoding diverse solutions from neural sequence models, 2018.

[8] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

[9] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.