Insert title here

Student 1 Student 2 Student 3

Abstract

Place your main findings here.

1. Introduction

2. Methodology

We start our task by first creating a global config that defines important parameters of our task. This is given in **Table 1**

Parameter	Value
model_name	SmolLM2-135M-SFT-Only
tokenizer_name	SmolLM2-135M-SFT-Only
dataset_name	yahma/alpaca-cleaned
test_size	400
max_new_tokens	128
repeat_ngram_block	3
repeat_threshold	4
reward_model	OpenAssistant/reward-model-deberta-v3-large-v2

Table 1. Global Parameters Used

Now, we start from the implementation of Greedy Search. We define our model and tokenizer using the $model\_name$ and $tokenizer\_name$ from **Table 1**. For the tokenizer, we ensure that pad tokens exist, and for the model, we resize its token embeddings in case there were special tokens. Then, using our $dataset\_name$ from **Table 1**, we load the dataset, shuffle it, and sample $test\_size$ (from **Table 1**) amount of items. Using each item, we construct a prompt, which is a dictionary containing id, prompt_text, instruction, and input, all of which are retrieved from the item. We also "sanitize" the prompt by removing any whitespace characters at the end of prompt string, and appending a sentinel ($\backslash nAnswer\ :\backslash$) at the end of this cleaned prompt. This is necessary to ensure that the language model generates a response.

Once this necessary setup is done, we start our greedy search by iterating over the prompts. For each prompt string, we pass it through our model once to store KV cache, and get logits for next token. Next, in our greedy generation loop, where we iterate over max_new_tokens, defined in**Table 1**, these logits are converted into probabilities using softmax and the next token Id is chosen as the one with highest probability (greedy). The log probability is saved and the chosen next token id is appended to the generated ids list. After this, first, we check if any repeated n-grams are of length $repeat\_ngram\_block$ (from **Table 1**), and then we check that if n-gram count is more than $repeat_threshold$ (from **Table 1**), generation is stopped. This avoids repeating the same phrases.

Moreover, we also check if the model is echoing prompt markers like "### Instruction" or "### Response". We do this by decoding the generated tokens, and if any such marker is found, then removing everything before that repeated marker. Lastly, we stop when EOS token is encountered. The token ids left at this point are decoded to get the generated text and average log probability of these is computed. After this, we perform a post-processing step that removes leading and trailing whitespace and cuts before the repeated markers defined, as we did in previously in our max_new_tokens loop. This is repeated for each prompt. Finally, using our results, we create a box plot showing distribution of average log-probability for each generation sample with the mean and sample count labeled.

Now we move to the Beam Search. For this methodology we proceed the same way as Greedy search by creating our setup, and iterating over prompts. However, instead of keeping one sequence, we maintain $beam\_width = 4$ sequences. When iterating over max_new_tokens, we compute the cumulative score as the running sum of log probabilities of selected next-token logits across each beam. The token IDs against the top-k or top-4 cumulative score is saved during a beam. After this beam iteration finishes, we pick the best k candidates across all beams. We iterate over candidates with these token IDs, extend the parent beam with these token IDs, and determine if EOS token ID is reached. We also check for n-gram repetition similar to how we did in Greedy Search and then store the new beams.

After this loop, we computed next logits for the new beams that didn't end and update their KV cache. These new beams will now be used in the next iteration of max_new_tokens loop, and if all beams have ended, we stop early. Once we exit this this loop, the best beam is picked based on the score, and beams that ended (complete sequences) are preferred. At the end we have our post-processed generated text, beam width, best beam score, and average log probability stored. Using our results, we create a histogram that plots the cumulative beam log-probability score distribution against number of beams, in addition to the box plot showing distribution of average log-probability.

Moving on to Top-K Sampling, for this technique, we use a list of temperatures, $[0.2, 0.5, 0.8, 1.0, 1.2]$ and $k = 50$, and start by defining our model, tokenizer, and evaluation prompts (similar to previous implementations). For each temperature setting, we iterate over our complete evaluation prompts. For each prompt, we sanitize it and pass it through our model once to store KV cache, and get logits for next token. Then we iterate over max_new_tokens (given in **Table 1**), scaled the next token logits by temperature, applied softmax to get probabilities, picked the top k highest probability tokens, and renormalized them so that their probabilities sum up to 1. From these top-k token ids, we randomly sample one, and compute its log probability. The token id sampled and its log probability computed is saved. We also check for n-gram repetition using sliding window, as done previously. Next, this chosen token is fed back to the model, KV cache is updated, and logits are computed for the next step. Moreover, we also check if the model is echoing prompt markers (as done previously). Finally, when the chosen token is EOS token, we stop the max_new_tokens iteration, post-process our generated text, and also compute average log probability of chosen token. This is repeated for each evaluation prompt.

After this, we use the final list of generated text for each prompt to compute Distinct-N score - unique n-grams / total n-grams - for different n-gram sizes ($N = 1, 2, 3$), both across prompts and within prompts. For across prompts, we generate one sample per prompt and aggregate all generated texts. Whereas, for within prompts Distinct-N, we generate 10 outputs from a single prompt. This measures the diversity of the generated text. Using these we only plot Distinct-N (for $N = 1, 2, 3$) across prompts for different temperature settings.

Moving on to Top-p Sampling, we use the same list of temperatures as Top-K sampling, and the cumulative probability threshold $p = 0.9$. The only difference in implementation for this compared to Top-K Sampling is that during iteration over max_new_tokens, we first sort the probabilities of the next token logits in descending order, and then find the first index where cumulative probability sum exceeds p. These tokens indices and probabilities are saved and then renormalized such that their sum equals 1. Using its results, we create a bar chart of Distinct-N (for $N = 1, 2, 3$) comparing across prompts and within-prompt for temperature=0.8.

In addition to this, we also run a separate Top-P and Top-K sampling against the temperature list defined. But in this implementation, we score each generation using the $reward\_model$ defined in **Table 1**. This is necessary for our scatter plot showing tradeoff between diversity (Distinct-1) and quality (reward model score) across temperature settings defined, for both Top-P and Top-K on the same plot. Finally, we also plot the average log probability of both Top-P and Top-K on the same plot against the temperature settings defined.

Now, we move on to our first alignment technique, called Direct Preference Optimization (DPO). The important parameters of our task are given in **Table 2**.

Parameter	**Value**
ref_model	SmolLM2-135M-SFT-Only
tokenizer_name	SmolLM2-135M-SFT-Only
dataset_name	HuggingFaceH4/orca_dpo_pairs
train_epochs	1
steps	500
batch_size	4
learning_rate	$2e-5$

Table 2. Global Parameters Used in DPO

We start by loading our dataset (given in **Table 2**) and normalizing the examples in it. This is done by finding a preference pair in the example and then converting it into a canonical dict in the form $\{prompt, chosen, rejected\}$. After this, the data is split into 80% training and 20% validation data. Then, we initialize our model and tokenizer (given in **Table 2**) similar to how we did in the previous part. Using this model, we create our reference model and freezing all its parameters, and prepare it for LoRA finetuning. This is done by first ensuring that the model is k-bit compatible, and then attaching LoRA adapters to it with $rank = 8$, $dropout = 0.05$, and $scaling\ (alpha) = 16$ to create our Parameter-Efficient Finetuned Model (PEFT) .

Then, we construct our DPO trainer. For this, we first define a config using the $train\_epochs$, $steps$, $batch\_size$, and $learining\_rate$. Next, we instantiate our trainer using the reference model, PEFT model, tokenizer, training and evaluation datasets, and the config. This trainer is then used to execute training on the training dataset, and the weights of the PEFT model are saved when training finishes. Move on to evaluation, we first compute verbosity bias by sampling first 500 prompts from validation dataset. We first compute mean, median, standard deviation, and skew of the distribution. Then, using a simple classifier, we group the prompts into categories (factual, explanation, and other), and compute all of the above metrics for each category as

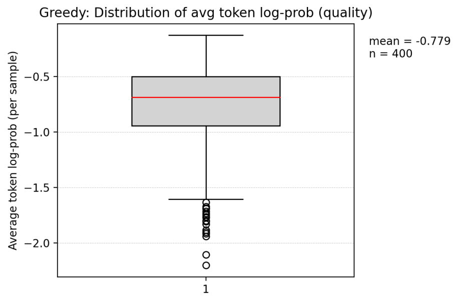

Figure 1. Average Log Probability of Each Generation Sample for Greedy Search With Mean and Sample Size Labeled

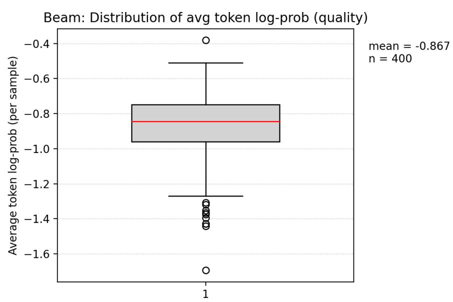

Figure 2. Average Log Probability of Each Generation Sample for Beam Search With Mean and Sample Size Labeled

well. Finally, we also check if prefixing the prompts result in generated responses of differing length (length limit = 50 words). For this, the compliance rate, and mean and std of deviations is computed. This is done for both reference model and PEFT aligned model. Then, using these result, for every stratified prompt category within a single model, we create a grouped boxplot that displays the distribution of response token lengths.

We also compute perplexity and mean token KL divergence by sampling first 200 prompts. This done by taking exponential of the negative fraction of the accumulated (across prompts) next token logits log probability and the number of next token prediction. Lastly, to investigate reward hacking, we use first create prompt-response pairs. The PEFT aligned modeis used as a surrogate to score both original and perturbed outputs after applying perturbations such as adding fillers, injecting keywords, or rearranging sentences. Standard deviation, fraction of positive deltas, and mean score change are calculated. Additionally, to show how the reward model reacts to surface-level perturbations by displaying both the average score change and the variability across individual responses.

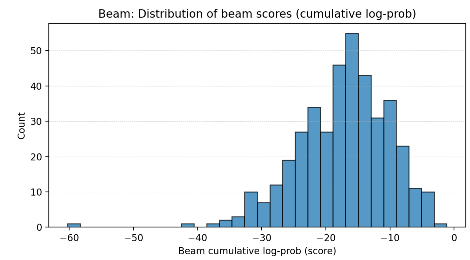

Figure 3. Distribution of Cumulative Beam Log Probability Score Against Number of Beams

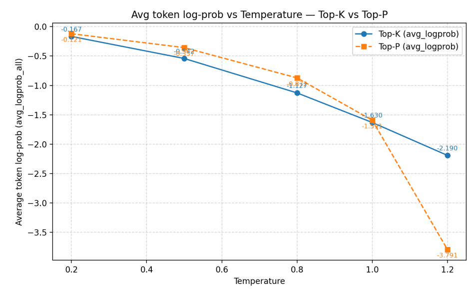

Figure 4. Average Log Probability Across All Prompts for Different Temperature for Both Top-P and Top-K

3. Results

For Greedy Search, the box plot of average log probability for each generation sample with mean and number of samples labeled is given in **Figure 1**.

For the Beam Search results, the average log probability box plot is provided in **Figure 2**. In addition to this, **Figure 3** shows the distribution of cumulative beam log-probability score. The plot of how average log probability changes with different temperature settings for both Top-K and Top-P is provided in **Figure 4**.

We also plot how Distinct-N (for $N = 1, 2, 3$) across prompts is affected by different temperature. For Top-P, this is given in **Figure 5**

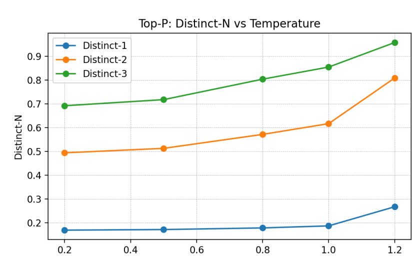

Figure 5. Top-K Distinct-N Across Prompts for Different Temperature

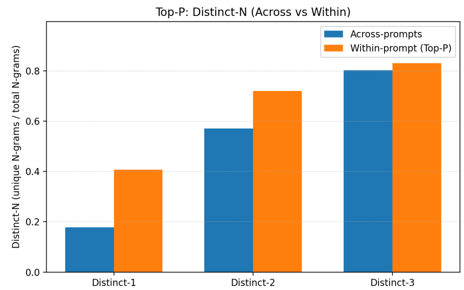

Figure 6. Top-P Distinct-N for Within and Across Prompts when Temperature=0.8

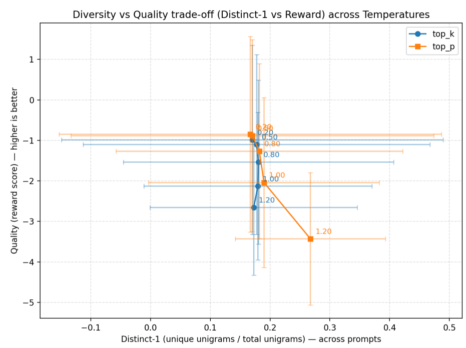

Figure 7. Diversity and Quality Tradeoff For Top-P and Top-K Across Temperature List Defined

In addition to this, the bar chart of Distinct-N for temperature=0.8, for both within prompts and across prompts is shown in **Figure 6**

For the quality vs diversity tradeoff for both Top-K and Top-P Sampling, we use Distinct-1 across prompts with a fixed $k = 50$ and $p = 0.9$ and across a range of temperatures (defined previously) and an open-source reward model, it is demonstrated by **Figure 7**

For our first alignment technique, DPO, the perplexity score of the PEFT aligned model is 11.00, while for the reference model is 10.23. From this, the mean token KL divergence is calculated to be 0.016. For verbosity bias, the metrics computed for both reference and aligned model is shown in **Table 3**.

Metric	**Aligned Model**	**Reference Model**
mean	66.25	67.70
median	59.0	60.0
std	37.54	38.04
skew	0.19	0.20
compliance rate	0.736	0.744
mean_deviation	17.56	18.15
std_deviation	12.34	12.48

Table 3. Metrics for Evaluating Verbosity Bias

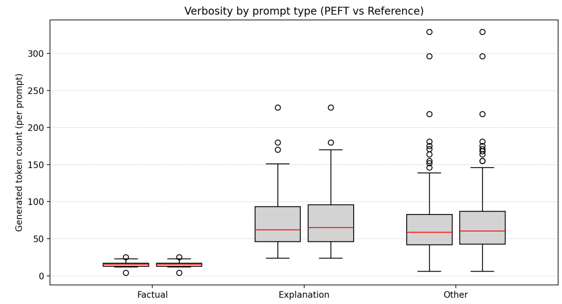

Figure 8. Distribution of Response Token Lengths for Each Prompt Category

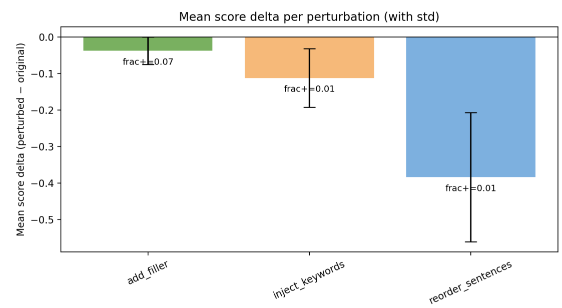

Figure 9. Change in Average Reward Score for Different Perturbations

Moreover, for each prompt category (factual, explanation, and other), the distribution of the generated response token lengths is shown in **Figure 8**.

Next, for the reward hacking part, the average changes in reward score for the PEFT aligned model under various perturbations is displayed in **Figure 9**.

Then, the aligned model's per-example reward score variability under the same perturbations is displayed in **Figure 10**.

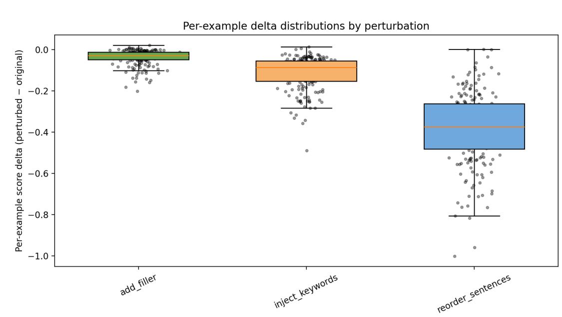

Figure 10. Variability in Reward Score, Per-Example, For Different Perturbations

4. Discussion

When comparing Greedy and Beam Search, Greedy gives a narrow range of average log probabilities across samples, since it always grabs the top probability token step by step. In contrast, beam search spreads out more, shifting the center a bit. This happens because beam may craft longer or richer endings, which sometimes lowers or keeps per-token log probability about the same if it lands on extended low-likelihood paths. This fits our expectation that greedy sticks close, wins locally, yet might skip smarter full-run options. On the other hand, beam pokes around, so its outputs swing wider in average log probability.

The cumulative beam log probability score demonstrates that most prompts bunch up in a middle range, while some stretch out toward higher total log probabilities. Since there's a wide space between the top and next-best beam, it suggests one clear winner path. So this graph measures how spiky or spread out the model's guesses are across whole outputs. Since we have strong spikes in our distribution, it means that the beam probably locks onto one main option. On the other hand, smoother shapes imply that multiple choices feel similarly right.

Top-K and also Top-P both get lower in average token log probability once temperature goes up. This is because higher temperature value spreads out the logits, adding noise while pulling down the average. This is in line with the expectation that higher temperature increases variety but weakens model's quality or how sure it feels (hence the lower average low probability). Moreover, in our results, Top-P plunges faster compared to Top-K that eases down relatively more gently. This could be because in Top-K, very low probability tokens can't enter the top-K even though temperature might be high. Whereas, in Top-P, the nucleus becomes much larger at higher temperature, so even low probability tokens are able to enter the candidate set.

Our results also show that in Distinct-N, the larger value of N climbs the fastest as the temperature is increased. This supports the hypothesis that, although the model uses a similar set of individual words (unigrams/Distinct-1) regardless of temperature, the sequences in which they are arranged (bigrams and trigrams) become significantly more distinctive and varied as temperature increases. It is not surprising that Distinct-3 has a higher score than Distinct-1 since it is more difficult to repeat a particular three-word phrase than a single common word like ”the” or ”is.”

For every N-gram, within-prompt diversity is constantly greater than across-prompt diversity. This suggests that generating multiple outputs for a single prompt forces the model to explore unique paths, resulting in high lexical diversity. On the other hand, responses to various prompts typically share more stopwords and structural language, which results in greater overlap and marginally lower distinctness scores. For both sampling techniques, increasing diversity (Distinct-1) typically results in a lower average reward score (quality).

But across similar levels of diversity, Top-P Sampling consistently achieves a higher reward score than Top-K Sampling, demonstrating its superior effectiveness. This benefit results from Top-P's capacity to modify its selection size in response to the peakiness of the distribution. This enables it to preserve competitive diversity while more successfully removing tokens that would lower the reward score. Therefore, nucleus sampling provides a better performance profile when striking a balance between output creativity and objective quality is crucial.

For our LLM Alignment task's DPO implementation, the PEFT-DPO adapter has a very low mean token KL (0.016) and a small alignment cost (perplexity ref = 10.23 vs PEFT = 11.00), which supports the conclusion that the adapter slightly increases perplexity while causing little deviation from the SFT policy. **Table 3**'s verbosity metrics (mean, median, std, and skew) are almost the same across models, and stratified behavior (longer explanation, short factual) is consistent with task expectations, supporting the idea that DPO was able to maintain appropriate response lengths.

Additionally, since the 50-word compliance and mean overrun are reasonable given token/word mismatches and natural variance in open-ended answers, the results can be justified as controlled, not pathological, verbosity. The negative mean deltas of the perturbation tests (add_filler, inject_keywords, reorder_sentences) and and per-example distributions concentrated at or below zero given in **Figure 4** and **Figure 5**, respectively, support the initial safety claims.

5. Conclusion

6. Contributions

References