PA5: RLHF

Hamza Habib Abdul Samad Rumaan Mujtaba github source

Abstract

This paper examinesins advanced methods for the creation, alignment, and interpretability of Large Language Models (LLMs). According to our evaluation of decoding strategies, Top-K sampling offers a worst balance between output diversity and quality than Top-P (nucleous) sampling, because Top-P sampling dynamically adjusts to the probability distribution. In our experiments, GRPO experienced catastrophic forgetting and mode collapse, frequently failing to produce meaningful output or falling victim to reward hacking. On the other hand, DPO maintained low perplexity and consistent response lengths. This demonstrates that Direct Preference Optimization (DPO) is much more stable and efficient for model alignment. Moreover, to test the "Platonic Representation Hypothesis.", we employ Universal Sparse Autoencoders (USAEs) in the mechanistic interpretability domain. We spotted two clear patterns in the features, showing ResNet and ViT keep their own basic traits but end up with similar advanced ideas. Because they share these higher-level representations, the rebuild quality takes a measurable hit - tied to how much they align.

1. Introduction

Despite achieving state-of-the-art results in producing fluent text, Large Language Models (LLMs) are still essentially limited by their pre-training goals. A model can frequently generates outputs that are toxic, biased, or inconsistent with human values, when it is trained only to maximize next-token likelihood. Additionally, these models function as opaque "black boxes," with polysemanticity hiding the internal mechanisms, such as how a model represents a concept like "curve" or "citation." The purpose of this report is to look into ways to address these two issues of interpretability and alignment. We start by examining several decoding strategies (Greedy, Beam Search, Top-K, and Top-P), In order to comprehend the trade-offs between generation quality and diversity. Then,we implement and compare the critic-free Group Relative Policy Optimization (GRPO) and the stability-focused Direct Preference Optimization (DPO), in order to assess the effectiveness of two alignment algorithms in guiding model behavior. Lastly, we reverse-engineer model activations using Universal Sparse Autoencoders (USAEs). We empirically test the "Platonic Representation Hypothesis" to determine whether different models independently discover the same statistical reality by training a common dictionary across various architectures.

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 (HuggingFace, 2024)
tokenizer_name	SmolLM2-135M-SFT-Only
dataset_name	yahma/alpaca-cleaned (Yahma, 2023)
test_size	400
max_new_tokens	128
repeat_ngram_block	3
repeat_threshold	4
reward_model	OpenAssistant/reward-model-deberta-v3-large-v2 (OpenAssistant, 2023)

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

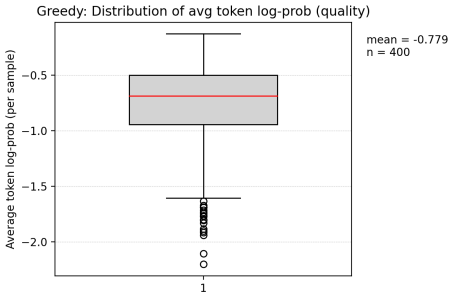

Figure 1. Average Log Probability of Each Generation Sample for Greedy Search With Mean and Sample Size Labeled

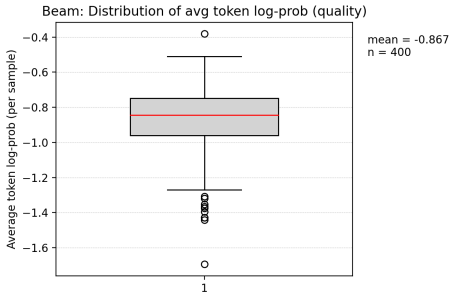

Figure 2. Average Log Probability of Each Generation Sample for Beam Search With Mean and Sample Size Labeled

against the temperature settings defined.

Now, we move on to our first alignment technique, called Direct Preference Optimization (DPO) (Rafailov et al., 2023). 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 (HuggingFaceH4, 2023)
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 (Hu et al., 2022) 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 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.

We utilized the ()SmolLM2-135M-SFT-Only model as the supervised fine tuned SFT baseline model. Due to its lightweight architecture, this model serves as an efficient testbed for analyzing alignment dynamics without being too much computationally expensive. To further optimize it for performance, all training phases employed LoRA and PEFT (Parameter-Efficient Fine-Tuning). Models were first loaded in 8-bit percision to reduce VRAM usage. After that, adapters were attached to the query and value projection layers with rank 16, and 8 for reward model. The alpha (α) was 32. Moreover, the implementation of these tasks relied heavily on Hugging Face, especially transformers, peft and trl.

The dataset used in this experiment was HuggingFaceH4/orca_dpo_pairs dataset. It consists of tuples of instruction; a prompt, a chosen response and a rejected response. We splitted dataset into 90% and 10%. The former was used to monitor overfitting and generalization, i.e. validation, while the later was 90% was used for training. For tokenization, we used SmolLM2 tokenizer, which ensured that left padding was applied during generation tasks to support inference.

We implemented GPRO (Shao et al., 2024), which eliminates the need for separate critic model by normalizing rewards within a group of sampled generations.

For each prompt, the policy sampled a group of 4 completions $\{y_1, y_2, y_3, y_4\}$. The reward model assigned scores to each completion. We calculated advantage A_i for the i-th completion which computed using this

Task 3 begins by importing the required libraries and setting random seeds so the experiment can be performed again. Two different pre-trained model models, ResNet-18 (512-dim features) and ViT-B/16 (Dosovitskiy et al., 2021) (768-dim features) are used. To allow fair comparison, we collect activations from the CIFAR-10 dataset (resized to 224 by 224) and normalize them to have unit variance and zero mean. This prevents one model with the larger values from overpower the other during training.

We define the Universal Sparse Autoencoder (USAE) architecture, which uses separate encoders and decoders (Bricken et al., 2023).for each model which connect through a single latent bottleneck Z with dimension m equal to 2048 (m much greater than d). The encoder maps input features to the latent space and uses a Top-K (k equal to 32) activation function to only keep the strongest features. The decoders then attempt to reconstruct the original activations from these shared sparse features.

For training, an AdamW optimizer with a learning rate of 0.001 is used. We train for 30 epochs and a batch size of 256 is used. In each step, we select a source model (A or B) at random, encode its input to Z, and then try to reconstruct the activations for both models. The error is calculated using the Mean Squared Error (MSE) between the normalized activations and their reconstructions across both models. The cross-reconstruction forces the shared bottleneck to learn features that can be interpretted by both ResNet and ViT.

For evaluation, we first compute the R square reconstruction scores to see how well the model can perform self-reconstruction and cross-reconstruction. Universality is measured using Normalized Firing Entropy, which tells if a feature is interprettable by both models (Entropy approximately equal to 1) or only by one (Entropy approximately equal to 0). Co-Firing Proportion is calculated to see how often both the models use the same feature for a certain image.

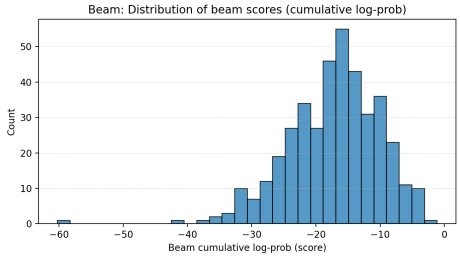

Figure 3. Distribution of Cumulative Beam Log Probability Score Against Number of Beams

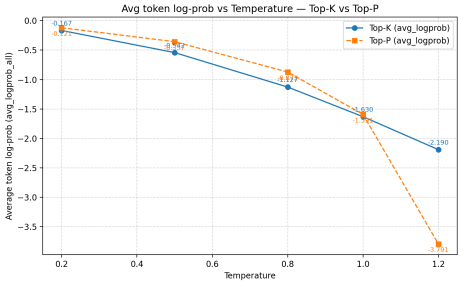

Figure 4. Average Log Probability Across All Prompts for Different Temperature for Both Top-P and Top-K

Finally, visual and quantitative analysis is performed. We use CAM with Gaussian blur regularization to visualize specific features, to check if the models visually see the same thing. To calculate the cost of sharing these features, we train an Independent SAE only on Model A using the same settings and compare its reconstruction performance to the USAE, referring to the difference as the "Alignment Tax."

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**

In addition to this, the bar chart of Distinct-N for temper-

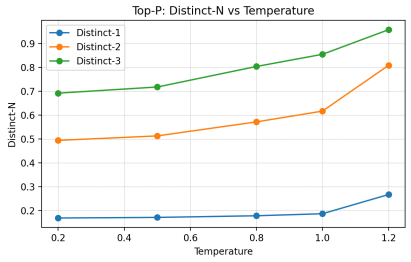

Figure 5. Top-K Distinct-N Across Prompts for Different Temperature

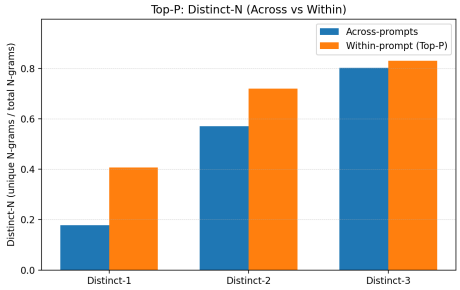

Figure 6. Top-P Distinct-N for Within and Across Prompts when Temperature=0.8

ature=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

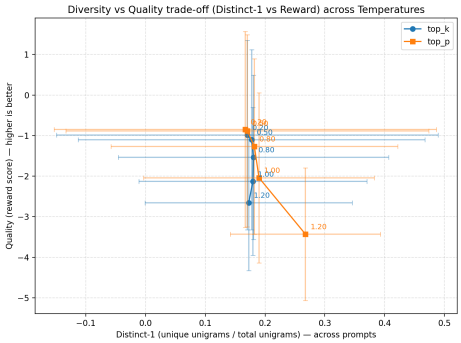

Figure 7. Diversity and Quality Tradeoff For Top-P and Top-K Across Temperature List Defined

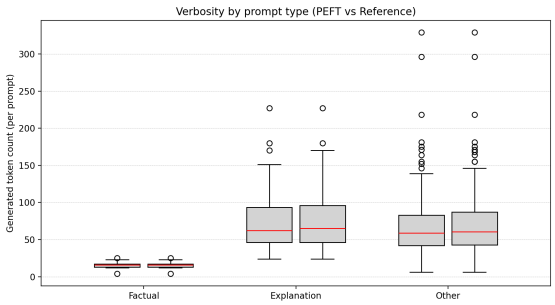

Figure 8. Distribution of Response Token Lengths for Each Prompt Category

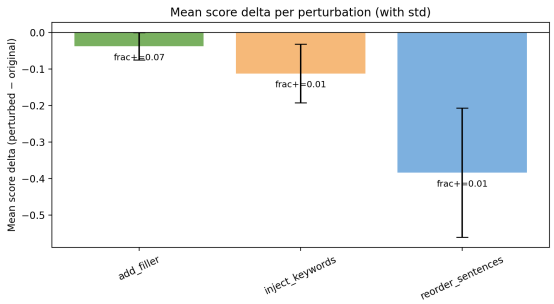

Figure 9. Change in Average Reward Score for Different Perturbations

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

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**.

We evaluated the Group Relative Policy Optimization (GPRO) model on 3 different axes: catastrophic forgetting, verbosity bias, and robustness to reward hacking. Results

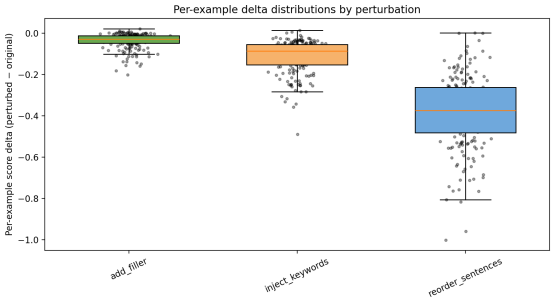

Figure 10. Variability in Reward Score, Per-Example, For Different Perturbations

are summarized in Table 4

Metric	GRPO Value
Catastrophic Forgetting	
Mean KL Divergence	0.0035
Perplexity	48.13
Verbosity Bias	
Mean Length	43.1
Median Length	0.0
Standard Deviation	48.3
Reward Hacking	
Normal Score	-3.471
Hacked Score	-2.357
Score Delta	+1.113

Table 4. Quantitative Results for GRPO Alignment Evaluation

The model achieved an average KL divergence of 0.0035, indicating a low drift from the reference model. However, the perplexity on the evaluation set was 48.13. The model also showed high variance in verbosity. The mean response length was 43.1 tokens, with a standard deviation of 48.3. For the reward hacking perturbation test, the model scored -3.471 to normal response and -2.357 to hacked response. This resulted in a differnce of +1.113 warning that model prefers the hacked responses.

We trained a Universal Sparse Autoencoder (USAE) with a latent dimension of m equal to 2048 using two distinct architectures, ResNet-18 (Model A) and ViT-B (Model B) with their normalised activations. The final reconstruction loss recorded was 0.587 after 30 epochs. We calculated the R square reconstruction scores for both self-reconstruction and cross-reconstruction to assess USAE performance. The results are recorded in Table 1. The cross reconstruction scores (0.618 and 0.718) demonstrate that the shared bottleneck Z was able to successfully learn features that both models could interpret.

We calculated the Normalized Firing Entropy for all 2048 features, to see which features were actually shared, this

Table 5. R^2 **Reconstruction Scores**

Source Input	Target: ResNet-18	Target: ViT-B
ResNet-18	0.759	0.618
ViT-B	0.718	0.771

is illustrated in Figure 2. The normalized training gave a distinct bimodal distribution, unlike the initial training on data that was not normalised. There were 763 Universal features with Firing Entropy higher than 0.9 and 319 model specific features with Firing Entropy less than 0.1. The Co-Firing proportion was calculated as 21.44 percent, showing that both models frquently agreed on which feature to use for a certain image.

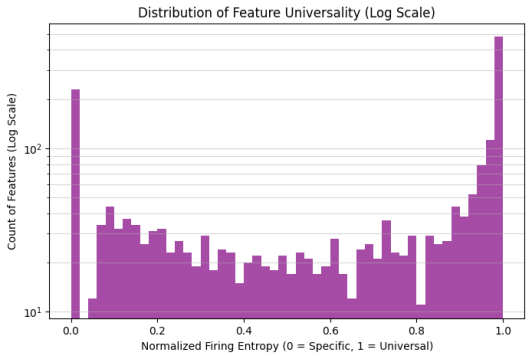

Figure 11. **Distribution of Feature Universality.** The histogram shows a clear split (bimodal). The peak at 1.0 represents shared features among the both models (Universal). The peak at 0.0 represents features used only by one model .(Specific).

We used a CAM visualisation to see the shared features among the models. Figure 2 shows the optimization results for Feature 1496 (high entropy feature) and Figure 3 for Feature 3 (low entropy feature). Feature 1496 shows very similar paterns for both models, confirming a "Platonic" representation. However, Feature 3 produces a clear image for Model A but only random noise for Model B, confirming "Feature Splitting."

Finally, to confirm that universality affects performance negatively or not, we compared the reconstruction quality of the USAE against an Independent SAE trained only on Model A. The drop in R square score when forcing the feature space to be shared is called the "Alignment Tax". The results are recorded in Table 2.

Table 6. **Alignment Tax Calculation (Model A).**

Method	R^2 Score	Alignment Tax
Independent SAE	0.8089	—
Universal SAE	0.7566	0.0523 (5.23%)

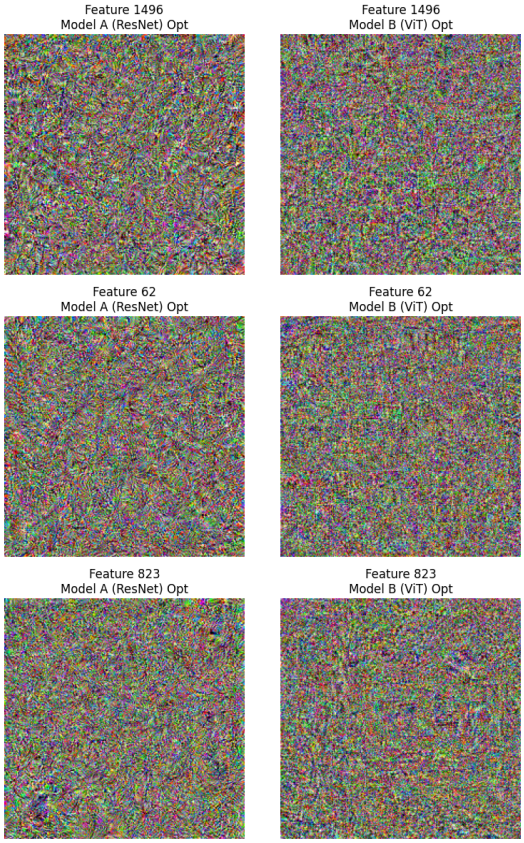

Figure 12. **Visualizing Consensus (Universal Feature 1496).** Both models generate similar patterns for this feature

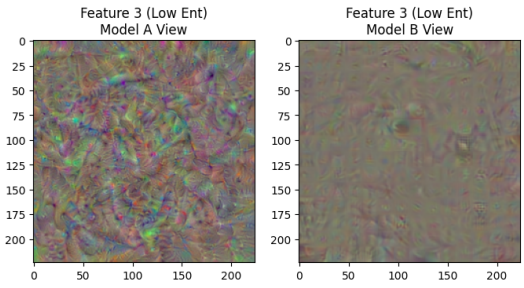

Figure 13. **Visualizing Divergence (Model-Specific Feature 3).** Model A produces a clear patter, Model B produces only noise

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.

Throughout this experiment, GPRO implementation exhibited significant instability and degradation of general capabilities compared to the baseline. The most alarming result was the median response length of 0 tokens. This shows a mode collapse where for 50% of the prompts, model just generated an end of sequence token. In other words, model failed to produce meaningful response. This indicates that policy might have found a way that saying nothing was safer strategy to avoid penalty from the judge model, or learning head destabilized the generation head. The high variance in token length suggests that the model oscillates between complete silence and moderately long responses.

There is a huge discrepancy between the extremely low KL divergence (0.0035) from reference model and the high perplexity (48.13). While the low KL divergence indicates that the policy model did not drift too far from the reference model, high perplexity suggests that model is confused and is struggling when it comes to predict evaluation text. This suggests that GPRO significantly impaired the model's general language modeling capabilities.

The model failed the robustness test when it comes to reward hacking. The perturbed response, like "I am a helpful AI" which was injected as a safety filler received a significantly higher reward (-2.357)than the correct factual response (-3.471). This confirms that GPRO successfully exploited the reward model's reliance on safety cues rather than semantic quality.

When comparing with DPO model, the DPO model was more stable and maintained a perplexxity of 11.00 close to the reference model (10.23) whereas the GPRO's perplexity spiked to 48. DPO showed a healthy mean length of 66.25 and a median of 59.0, demonstrating consistency in its responses. In contrast, GPRO preferred to stay silent to avoid negative penalties.

We observed an "Alignment Tax" of 5.23 percent when comparing the Independent SAE (R square equal to 0.8089) to the Universal SAE (R square equal to 0.7566). This drop happens because the model is forced to use a shared bottleneck between the Convolutional ResNet and the Transformer ViT models. However,since we retained approximately 93 percent of the original quality. This suggests the tax is worth the ability to interpret two models simultaneously.

There were 763 shared features and 319 unique features, this confirms feature splitting. Since the two models have different input sizes and our SAE bottleneck (m equal to 2048) was sufficiently large, the SAE was able to keep some features distinct. Instead, the SAE assigned some specific neurons to ResNet only and some to ViT only in order to handle the dimensionality mismatch.

We wanted to see if these "Model-Specific" features were just noise, however Figure shows a clear pattern for ResNet but only random noise for ViT. This suggests that unique features are useful for the specific architecture (ResNet requires them, though ViT does not). This suggests the "Platonic Representation Hypothesis" is only partially correct, as models agree on high-level concepts, but keep low-level architectural tools distinct.

We observed a low alignment score (Co-Firing Proportion of 21.44 percent). Since the original paper got higher alignment using larger models, our results suggest universality is an emergent property of scale. Larger models trained on more data are forced to learn the true underlying pattern of data, while smaller models are prone to learning spurious correlations specific to their architecture.

5. Conclusion

Top-P sampling was observed that it performs better than Top-K when it comes to reward scores with consistency. This is also observed by decoding strategy experiments where the removal of low probability tokens while maintaining diversity is done. In the experiments, DPO and GRPO had a huge performance gap as observed in the experiments.

GPRO suffered from extreme instability while DPO had a consistent training progression. The use of Universal Sparse Autoencoders supports the interpretation of Platonic Characteristics described as "universal" which is common in vision transformers and resnet model architectures. The experiments showed that while models converge to closely similar absolute truths. Their internal representation of information is still significantly impacted by the characteristics of their architectures and its constraints. These results specifically included the preservation of model-specific features and the existence of tax on alignment, evident by a 52.3

6. Contributions

References

Bricken, T., Templeton, A., Batson, J., Chen, B., Jermyn, A., Li, T., et al. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. URL https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

HuggingFace. Smollm2-135m-sft-only. https://huggingface.co/HuggingFaceTB/SmolLM2-135M-SFT-Only, 2024. Accessed: 2025-01-01.

HuggingFaceH4. Orca dpo pairs dataset. https://huggingface.co/datasets/HuggingFaceH4/orca_dpo_pairs, 2023. Accessed: 2025-01-01.

OpenAssistant. Openassistant reward model DeBERTa-v3-large-v2. https://huggingface.co/OpenAssistant/reward-model-deberta-v3-large-v2, 2023. Accessed: 2025-01-01.

Rafailov, R., Sharma, A., Mitchell, E., Manning, C. D., Ermon, S., and Finn, C. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, A., Xiao, M., Li, Y., Zhang, Y., et al. Deepseek-math: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Yahma. Alpaca cleaned dataset. https://huggingface.co/datasets/yahma/alpaca-cleaned, 2023. Accessed: 2025-01-01.