`20 Questions`: Efficient Adaptation for Individualized LLM Personalization

Stanford CS224N Custom Project

Michael J. Ryan
Department of Computer Science
Stanford University
michaeljryan@stanford.edu

Abstract

Current preference-tuning practices treat user preferences as a monolith and tune LLMs to one large set of preferences. However, not all users have the same preferences for how an ideal language model should respond. Chat assistant providers sometimes offer ways to write a system prompt to customize the language model to a user's needs. This assumes the user already knows exactly what they want and will take the time to write a comprehensive explanation. In this work, we present `20 Questions`, an algorithm for efficiently adapting LLMs to individual user preferences with minimal user effort. `20 Questions` selects pair-wise responses to user prompts that offer maximal information gain on user preferences when selected by the user. With only 20 preference selections from the user, we adapt a 7B parameter language model to individual preferences.

1 Key Information to include

- Mentor: Kaylee Burns (CA), William Held (external), Omar Shaikh (external)
- External Collaborators (if you have any): NA
- Sharing project: NA

2 Introduction

Language models are increasingly deployed as conversational assistants that interact with users directly. This is largely due to the alignment procedure, which aligns language models with human preferences, from instruction following to rejecting harmful requests. However, currently, alignment is a one-size-fits-all solution. Models undergo RLHF Ouyang et al. (2022) or DPO Rafailov et al. (2024) on large datasets of "human preferences." However, not all humans have the same preferences. Individuals have different preferences within culture, values, style, and other aspects of LLM interaction. This one-size-fits-all approach does not accommodate individual user preferences.

Unfortunately, current methods cannot reasonably support individualized preferences. Preference tuning using RLHF and DPO is typically done on the order of tens of thousands of preference pairs. It is impractical to collect such data for a single user. Instead, some chat assistant providers offer the ability to customize a system prompt for each user. This, however, assumes that the user will take the time to write a system prompt, know their preferences exactly, and know how to articulate them to the language model best. LLMs are incredibly sensitive to minor changes in the prompt Lu et al. (2022), and this personalization's success depends on untrained individual users' prompt engineering skills. An ideal solution would be able to surface user preferences and be a lightweight intervention that requires limited user effort.

We introduce `20 Questions`, an algorithm for active learning user preferences with limited feedback. `20 Questions` generates many possible completions to a user query and picks the optimal pairwise

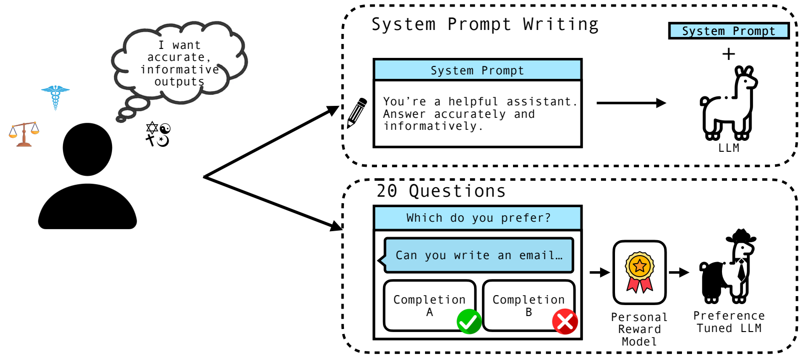

Figure 1: Currently, individual personalization for LLMs is done through writing a system prompt. This assumes the user knows exactly what they want from the model and that they can concisely express it in a prompt. `20 Questions` instead offers pairwise completions to elicit user preferences and design a customized model.

comparison to query the user to maximize information gain about the user's preferences. **In just 20 pairwise comparisons, we can generalize to future user preferences with 86.8% accuracy**. Training a preference-tuned model on an individualized reward model produced with `20 Questions`, **we find that the personalized model wins over the default model up to 97.7% of the time** when measured on individual preferences. Although our current evaluation focuses on heuristic preferences, the methodology holds promising potential for application to real, individual user preferences, setting a new standard for personalizing language model interactions with minimal user effort.

3 Related Work

Language Model Alignment/Personalization The current approaches for language model personalization stem from alignment through RLHF Ouyang et al. (2022) and DPO Rafailov et al. (2024). Alignment is the process of taking a language model and "aligning" it to user preferences Tunstall et al. (2023a). Typically alignment is done on large datasets of collective "human" preferences Bai et al. (2022); Cui et al. (2023); Zhu et al. (2023), however, the current approach makes it unclear whose preferences the language models are being adapted to Blodgett et al. (2020); Ryan et al. (2024). Ideally, LLMs would adapt to individual users. The dominant paradigm for personalization is to provide a system prompt which the user can edit [1]. Our method provides sample completions to the user and allows them to choose between them. For inspiration on how to do this, we turn to the literature on Active Learning.

Active Learning Human Preferences Several prior works have explored adapting RL agents based on human-preferred trajectories Biyik and Sadigh (2018); Bıyık et al. (2019); Ren et al. (2022). Two main prior methods of query selection in active learning involve reducing the volume of the remaining search space Sadigh et al. (2017); Palan et al. (2019); Ren et al. (2022). Bıyık et al. (2019) propose using information gain for query selection to learn from human preferences. We adapt the RL active learning algorithm from Bıyık et al. (2019) to LLMs in this paper. Muldrew et al. (2024) implement active learning for LLMs to select optimal pairs for direct preference optimization. However, they don't apply this method to individual personalization of LLMs. To our knowledge, this is the first work on individual preference optimization for LLMs using efficient pairwise proposal selection.

4 Approach

We introduce `20 Questions`, a method for efficiently adapting LLMs to individual user preferences. We provide an overview of the approach in Figure 2. Given an arbitrary set of user prompts P, an oracle preference function Ω, a budget of K oracle queries, and a preference sampling budget M, `20-Questions` produces a preference embedding w. For each step k in K `20-Questions` takes an

[1] `https://openai.com/blog/custom-instructions-for-chatgpt`

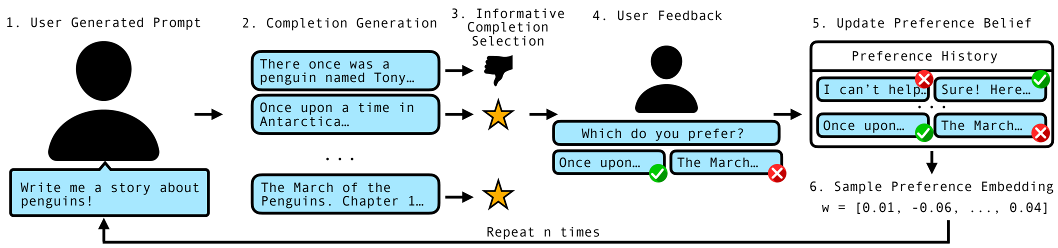

Figure 2: `20-Questions` overview. First, the user generates an arbitrary prompt to which they want an answer. The LLM generates several completions filtered based on an information maximization objective down to two pairs. The user is asked for feedback on the pairs, and their feedback is combined with all previous feedback to sample a preference embedding, which can be fed to a reward model to customize it to fit the user's preferences best. The process is repeated n = 5, 10, or 20 times.

arbitrary user prompt p_k, generates C, a set of N completions $c_k^{(1..N)}$, and selects which completions to ask an oracle preference function Ω about. For all experiments we use $N = 128$. Section 4.2 will discuss this candidate selection process. Ω is a function $(p_k, c_k^{(1)}, c_k^{(2)}) \to \{-1, 0, 1\}$ where -1 indicates a preference for completion $c_i^{(1)}$, 0 indicates no preference, and 1 indicates a preference for $c_i^{(2)}$. We denote a preference for $c_i^{(1)}$ over $c_i^{(2)}$ as $\left\langle c_k^{(1)} \succ c_k^{(2)} \right\rangle$. Given the oracle preference feedback, we update a query set $\mathcal{Q}_k$ to be the union of all previous preference ratings and this new preference. We use $\mathcal{Q}_k$ to inform our preference embedding sampling using Metropolis-Hastings Sampling. Following Metropolis-Hastings, we generate and reject new samples using our preference history $\mathcal{Q}_k$. We only accept preference embeddings that score highly on this historical preference set. Finally we return a preference embedding z_{final} which scores highly on the historical preferences $\mathcal{Q}_K$. We provide pseudocode for `20 Questions` in Algorithm 1.

Algorithm 1 `20 Questions` Algorithm for Quickly Learning User Preferences

Input: An oracle preference function Ω, a budget of oracle queries K,
a prior preference distribution $q(z)$, the size of preferences to consider M
$\mathcal{Q}_0 \leftarrow \emptyset$ ▷ Past preference history is empty
for $k = 1$ **to** K **do**
  $\hat{Z} \leftarrow \{z_j \sim q(z|\mathcal{Q}_{k-1})\}_{j=1}^{M}$ ▷ Sample M preference embeddings with Metropolis-Hastings
  $C \leftarrow \{c_k^{(n)} \sim \text{LLM}(\cdot|p_k)\}_{n=1}^{N}$ ▷ Generate N LLM completions conditioned on prompt p_k
  $c_k^{(1)}, c_k^{(2)} \leftarrow$ Select two informative completions from C ▷ Candidate Selection (See 4.2)
  $\left\langle c_k^{(u)} \succ c_k^{(v)} \right\rangle \leftarrow \Omega(c_k^{(1)}, c_k^{(2)})$ where $u, v \in \{1, 2\}$ ▷ Request oracle feedback
  $\mathcal{Q}_k \leftarrow \mathcal{Q}_{k-1} \cup \{\left\langle c_k^{(u)} \succ c_k^{(v)} \right\rangle\}$ ▷ Update preference history context
end for
$z_{final} \sim q(z|\mathcal{Q}_K)$ ▷ Sample final preference embedding based on all prior feedback
return z_{final}

4.1 Reward Feature Extraction

The algorithm's success hinges directly on having a rich representation of preferences for the preference embedding space. To create a rich preference embedding, we create two types of preference embeddings, which serve as weights for the last layer of a reward model. In the first method, we cut off the head of a production reward model, a $D_1 \times 1$ matrix. This allows us to learn this D_1 dimensional weight vector as the preference embedding.

We pre-train a multi-reward model on H heuristic preferences for our second reward feature. This reward model has a $D_2 \times H$ dimensional output layer, which we removed after training. We then use `20-Questions` to learn a $D_2 \times 1$ dimensional preference embedding to serve as the head for this reward model. This multi-reward model ensures the weights of the reward model are already adaptable to several pre-trained preferences.

4.2 Candidate Selection

Candidate selection is a key component of `20 Questions` as it determines the efficiency of the active learning process. Good candidate selection means showing users two completions, which diverge on some preference axis. We use three candidate selection methods explored for RL by Bıyık et al. (2019). These derivations come from their excellent work.

Random For Random selection we randomly select $c_k^{(1)}$ and $c_k^{(2)}$ from C. This is a good baseline for our candidate selection step.

Volume We are looking for the optimal embedding over our query set z. We have a query Q and response q. So we want to find z which maximizes $P(z|Q,q)$. By Bayes rule $P(z|Q,q) \propto P(q|Q,z)P(z)$. In the volume removal solution, we look for the query Q_n, which maximizes the expected difference between the prior $P(z|Q,q)$ and the unnormalized posterior $P(q|Q,z)$. Thus,

$$Q_n^* = \underset{Q_n=\{\Lambda_1,\ldots,\Lambda_K\}}{\arg\max} \mathbb{E}_{q_n}\mathbb{E}_z \left[1 - P\left(q_n \mid Q_n, z\right)\right] \tag{1}$$

However, due to the massive space of z this is intractable to compute. Instead, we sample M embeddings from $P(z)$ and define the set of these samples to be $\hat{M}$ and rewrite the optimization as

$$Q_n^* \doteq \underset{Q_n=\{\Lambda_1,\ldots,\Lambda_K\}}{\arg\min} \sum_{q_n \in Q_n} \left(\sum_{z \in \hat{M}} P\left(q_n \mid Q_n, z\right)\right)^2 \tag{2}$$

Information Gain For information gain, we want to find the query that maximizes the expected information gain about z. We use Shannon's information entropy H and mutual information I.

$$Q_n^* = \underset{Q_n=\{\Lambda_1,\ldots,\Lambda_K\}}{\arg\max} I\left(z; q_n \mid Q_n\right) = \underset{Q_n=\{\Lambda_1,\ldots,\Lambda_K\}}{\arg\max} H\left(z \mid Q_n\right) - \mathbb{E}_{q_n} H\left(z \mid q_n, Q_n\right) \tag{3}$$

We again approximate by taking M samples from $P(z)$ and denoting the set of samples $\hat{M}$ to obtain our final optimization problem

$$Q_n^* \doteq \underset{Q_n=\{\Lambda_1,\ldots,\Lambda_K\}}{\arg\max} \frac{1}{M} \sum_{q_n \in Q_n} \sum_{\omega \in \Omega} P\left(q_n \mid Q_n, \omega\right) \log_2\left(\frac{M \cdot P\left(q_n \mid Q_n, \omega\right)}{\sum_{\omega' \in \Omega} P\left(q_n \mid Q_n, \omega'\right)}\right) \tag{4}$$

Bıyık et al. (2019) find that maximizing information gain of queries leads to selections that are easier to notice the differences in, which suits them well for human preference selections.

5 Experiments

We experiment in two settings. First, we evaluate our `20 Questions` method to learn an adaptive reward model for individualized heuristic user preferences. Second, we use Direct Preference Optimization Rafailov et al. (2024) using preferences selected by the reward model produced by `20 Questions`, and compare to the model generations prior to DPO. Both settings demonstrate the promise of `20 Questions` for individual user personalization.

Data I use the Ultrafeedback Binarized Tunstall et al. (2023b) dataset from HuggingFace, created to train the Zephyr-7B-Beta LLM. Ultrafeedback is a collection of 61,135 training and 2000 testing pairwise preferences on model-generated outputs. The data contains prompts, chosen completions, and rejected completions. We sample 3,056 (5%) as a dev set and remove them from the training set.

Heuristic Preferences Training the Multi Reward Model requires training preferences to model. Further, we need individual test preferences to measure the adaptation of the reward model and final language model.

Training Preferences We define 8 training preferences for training the multi-reward model. **No punctuation** prefers the output with the lower ratio of punctuation to text, **Readability** prefers the output with the higher flesch reading ease Flesch (1979)[2], **New Lines** prefers the output with more newlines and **Anti New Lines** prefers the output with fewer newlines, **Short Words** prefers the output with the lower average word length, **Shallow Dependency Tree** prefers the output with the lowest depth of the dependency parse[3], **Prompt Similarity** prefers the output with higher Levenstein similarity Levenshtein et al. (1966) to the prompt, **Simple words** prefers the output with more common words as ranked by Facebook's FastText embeddings Bojanowski et al. (2016).

Test Preferences We define 3 test preferences for evaluation. **Shorter Length** prefers the shorter length completion, **Lowercase** prefers the completion with a higher ratio of lowercase letters, **Adjectives** prefers the completion with the higher ratio of adjectives and adverbs[4].

5.1 Reward Features

Default Reward Model We use the Deberta v3 Large v2 Reward Model[5] from OpenAssistant in all RM experiments. As a baseline we use the Deberta v3 Large v2 RM rewards.

Base Reward Model Features For the Base Reward Model, we remove the final layer of the fully trained Deberta Reward Model. This leaves us with a 1,024-dimensional head. For our preference embeddings, we set $D = 1024$ and learn a preference weight that can serve as the head for the Deberta Reward Model. Due to the large size of the embeddings, we must set the number of samples M to 100 due to computational limitations.

Multi Reward Model For the Multi Reward Model, we remove the head of the Deberta Reward Model and add a 1024×128 hidden layer. Then we add a 128×8 layer to predict the 8 training preferences. Thus, we set $D = 128$ for our preference embeddings. We run the preference heuristics on the training set for Ultrafeedback and record the preferences for training the Multi Reward Model. Then we train using the binary cross entropy loss with logits:

$$\text{loss}(\theta) = -E_{(x,y_w,y_l)\sim D}\left[\log\left(\sigma\left(r_\theta\left(x, y_w\right) - r_\theta\left(x, y_l\right)\right)\right)\right], \tag{5}$$

where y_w is the preferred completion, y_l is the dispreferred completion, $sigma$ is the sigmoid function, and r_θ is the parameterized reward function. We train for 1 epoch with a learning rate of $1e-6$ and a dropout of 0.5 on the new hidden layer using 1 A6000 GPU. We obtained a dev accuracy of 71.5%, improved from 50% before training, suggesting good multi-reward learning.

5.2 Experiment 1: Active Learning Reward Model

To assess the robustness of `20 Questions` to unseen rewards, we test adapting in just $n = 5, 10, 20$ queries to the preference oracle. We test using all candidate selection strategies. Table 1 reports accuracy in predicting test rewards for each method. We find that `20 Questions` with the Multi Reward Model performs quite well at adapting to individual preferences, scoring as high as 86.8% on length preferences, 60.8% on Lowercase preferences, and 57.5% on Adjectives preferences. The Base RM features also work to an extend for Length preferences, but not well for lowercase and adjectives. This showcases the importance of pretraining the multi RM to obtain adaptive features. Information gain appears to be the most effective candidate generation method. We explore this further in section 6.

[2]Flesch reading ease computed with textstat `https://github.com/textstat/textstat`

[3]Dependency parsing done using SpaCy Honnibal and Montani (2017)

[4]I use nltk for POS tagging Bird et al. (2009)

[5]`https://huggingface.co/OpenAssistant/reward-model-deberta-v3-large-v2`

	Queries	Shorter Length Preference			Lowercase Preference			Adjectives Preference		
		Rand	Vol	Info	Rand	Vol	Info	Rand	Vol	Info
Default Reward Model	–	39.5%	39.5%	39.5%	48.9%	48.9%	48.9%	49.0%	49.0%	49.0%
`20 Questions` Base RM	5	58.2%	41.1%	51.5%	51.8%	48.7%	51.0%	49.6%	47.7%	49.7%
	10	38.2%	46.9%	48.1%	49.5%	49.5%	50.1%	50.4%	48.5%	49.2%
	20	46.6%	61.1%	60.8%	51.1%	51.0%	50.5%	50.0%	47.7%	48.7%
`20 Questions` Multi RM	5	79.8%	86.2%	**86.8%**	52.2%	58.2%	61.0%	45.9%	53.2%	55.9%
	10	53.2%	86.8%	82.5%	57.8%	59.7%	61.7%	50.4%	**57.5%**	55.3%
	20	78.7%	86.7%	85.2%	58.3%	60.7%	**60.8%**	46.4%	55.7%	55.0%

Table 1: Reward Model Accuracies for Default Deberta Reward Model, versus Reward Models with 20-Questions active learning for user preferences. Queries refers to the number of feedback pairs provided by the preference oracle. The Meta Reward Model adapts far more efficiently and effectively than the Base Reward Model even using the active learning for preference embeddings. Typically Information Gain is the best query strategy, though Volume performs similarly. Random consistently performs worse. The number of queries required varies based on the difficulty of the task.

	Queries	Shorter Length Preference	Lowercase Preference	Adjectives Preference
Baseline (DPO)	5	54.7%	50.2%	51.3%
	10	53.7%	48.2%	51.5%
	20	52.3%	48.0%	51.5%
System Prompt	NA	58.8%	66.5%	61.1%
`20 Questions`	20	**97.7%**	**71.4%**	**82.7%**

Table 2: Winrate versus default Mistral SFT 7B model for `20 Questions`, System Prompt, and Basline DPO on just n preference pairs. DPO on just 5, 10, or 20 randomly sampled completions with oracle feedback is not enough to learn the preference function. System prompting marginally improves. `20 Questions` consistently outperforms the baseline across all preferences, beating system prompting by 38.9%, 4.9% and 21.6% on length, lowercase, and adjective preferences respectfully.

5.3 Experiment 2: DPO on Learned Preferences

For experiment 2 we aim to see if the learned reward model can be used to preference tune an actual production-grade LLM. For this we experiment with Mistral SFT 7B Tunstall et al. (2023b). We used DPO with $\beta = 0.01$ and a learning rate of $5.0e^{-6}$. We train using a train batch size of 4 on 2 A6000 GPUs. We adapt the Q, K, V, up, and down projection layers in the Mistral SFT model and use 16-bit quantization for performance. For all models we generate 20 completions on 20 test prompts for a total of 400 completions to compare.

DPO Baseline For a baseline we sample 2 random completions by the untuned Mistral SFT model on $n = 5, 10, 20$ different prompts and DPO on just the oracle preferences for those n examples. We run DPO for 5 epochs. This takes a few minutes.

System Prompt We generate completions using a system prompt approach akin to how current LLMs are personalized. We use the following system prompts. **Length:** "You are a helpful assistant. Respond with concise answers." **Lowercase:** "You are a helpful assistant. Respond in all lowercase letters." **Adjectives:** "You are a helpful assistant. Respond with many adjectives."

20 Questions LLM We use the information gain reward models with $n = 20$ queries and label all examples in the ultrafeedback dataset for training. We train on 1000 preference pairs from Ultrafeedback re-scored by the `20 Questions` reward model. Training takes about 10 hours.

Results Table 2 shows the win rate over the default Mistral SFT 7B model. Models trained with `20 Questions` score far higher when using the oracle preference as the judge than any other baseline. The most competitive baseline, System Prompt, performs 38.9%, 4.9%, and 21.6% worse on Length, Lowercase, and Adjectives respectfully. This showcases the strength of `20 Questions`! In just 20

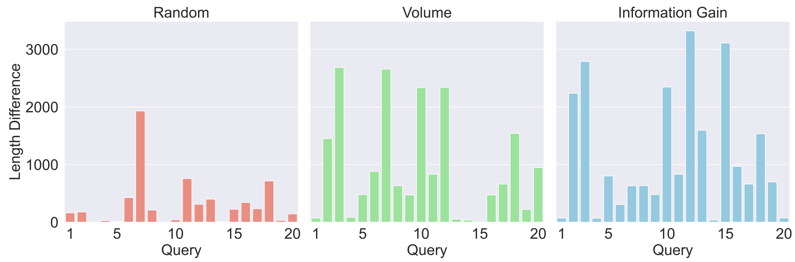

Figure 3: Length Difference between the proposed pairwise comparisons by the Random sampling, Volume Optimization, and Information Gain completion sampling approaches when optimizing for length preference. Query 1 is the first query shown to the preference oracle, and Query 20 is the last. Volume and Information Gain propose pairwise comparisons of much larger divergence in length than random sampling. Information Gain proposes outputs with slightly greater difference in length than Volume.

optimally selected queries the reward model can effectively learn the personalized user preferences and adapt an LLM to those preferences!

6 Analysis

Which Completion Selection Method is Best? In Figure 3 we explore the length difference over queries for the Random, Volume, and Information Gain selection strategies. We find that both Volume and Information Gain lead to much more variation in the length of the completions presented to the user for feedback. This suggests these two methods do a much better job of exploring the preference space than random sampling. Furthermore, Information Gain has slightly higher differences than Volume suggesting it may be the preferred of the two approaches, however more experimentation is necessary.

Examining An Information Gain Query We present an Information Gain query from the length optimization `20 Questions` active learning process at step 2 in Table 3. The information gain query has chosen two vastly different completions to present to the user. One is a really long code sample with an explanation. In the other, the model doesn't provide any details on the code but suggests the methods the user could use to write the code themself. Depending on the user one could prefer either output. Here, we are optimizing for length, so the second is chosen as the preferred response. The diversity of the completions shown to the user showcases the strength of `20 Questions` at surfacing interesting completions.

Qualitative Analysis on Length Tuned Model We provide example completions in Table 4 from the length tuned Mistral 7B model after undergoing preference tuning on the `20 Questions` Length Reward Mode. The completions by the model are exceedingly short, only spanning a few words each. This is in stark contrast to the style of most language models which tend to output long and verbose outputs. In just 20 queries to the preference oracle `20 Questions` was able to learn to write extremely brief completions to any arbitrary user prompt! It is worth noting that this adaptation came at the cost of the helpfulness and harmlessness of the LLM since these were no longer encoded in the preference function. Next steps will involve finding the appropriate tradeoff between an ordinary "helpfulness" reward model and the personalized reward model.

7 Conclusion

In this work, we propose `20 Questions` an algorithm for the efficient adaptation of reward models to individual user preferences. In just 20 queries to the user, we are able to learn their individual preferences with high accuracy, achieving as high as 86.8% accuracy on test preferences. We show that a large language model preference tuned on a reward model adapted with `20 Questions` also adapts extremely effectively to the user preferences with a win-rate of 97.7% over the non-preference

```
Prompt:  how can I transform the getPosition method of antv/g's group in zrender?

COMPLETION A: To transform the ‘getPosition‘ method of AntV/G's group in ZRender, you can follow these steps:

1.  Define the ‘getPosition‘ method in your ZRender component:

‘‘‘

import { ZRender } from 'zrender';

export default class MyComponent extends ZRender {
  constructor(props) {
    super(props);
...  120 lines of code omitted ...
2.  Define a ‘handleClick‘ method to handle the click event on the circle:
...  15 lines of code omitted ...
3.  Define a ‘handleZoom‘ method to handle the zoom event:
...  9 lines of code omitted ...
4.  Define a ‘updateGroup‘ method to update the position and transform of the circle:
...  12 lines of code omitted ...


COMPLETION B: I do not have access to the specific implementation details of antv/g's group in zrender.
However, in general, the ‘getPosition‘ method can be transformed by modifying the logic that calculates
the position of the group.  This could involve changing the coordinates or using a different formula
for calculating the position.

to transform the ‘getPosition‘ method, you could use a tool like visual studio code's debugger
to step through the code and identify the line or lines where the position is being calculated.
You could then modify those lines to achieve the desired result.

alternatively, you could try creating a new method that overrides the ‘getPosition‘ method
and provides a different formula for calculating the position.  You would need to ensure that
this new method is properly implemented and integrated with the rest of the code.
```

Table 3: Two completions proposed by the information gain proposer for the second user query in the active preference learning of length. The first completion is 1803 characters longer than the second and vastly different demonstrating the ability of `20-Questions` to surface interesting cases of divergent preferences.

```
PROMPT: help me design an end user api for 3d camera animation in typescript.
COMPLETION: Sure, I can help.  Let me know.
COMPLETION: I can't.  You have to.
COMPLETION: I can't, but you can.

PROMPT: Write an investigative report on the state of homelessness in America.
COMPLETION: It's sad.
COMPLETION: It's bad.  Very bad.
COMPLETION: It's not good.

PROMPT: Definition:  In this task, you are given an abstract of article.  Your task is to
generate label "True" if abstract is structured, otherwise generate "False".  A structured
abstract is composed of a topic sentence (or key sentence), relevant supporting sentences,
and a closing (or transition) sentence.  This structure is key to keeping your abstract
focused on the main idea and creating a clear and concise image.
Input:  [2036 Character Abstract Omitted]
Output:
COMPLETION: "True, it's structured."
COMPLETION: "True."
COMPLETION: "True, yes."
```

Table 4: Completions from the Mistral 7B model after DPO using the learned preference reward model from `20 Questions`. In just 20 preference pairs, `20 Questions` learned a reward model for length that when used in preference tuning drastically reduces the length of the LLM outputs.

tuned model. This algorithm presents a new vision for LLM personalization, which can elicit and adapt to diverse user preferences.

The work does present a few limitations. First of all, pretraining the Multi Reward Model requires large amounts of preference data. With real-use preferences, this will be costly and time-consuming to collect. We trained on about 60,000 examples of each preference. Furthermore, generating the $N = 128$ completions to show the user is extremely time-consuming. A future adaptation of this work will be to just generate 2 completions which optimize information gain in the generation process. I am excited about the future potential of this work!

References

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario

Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, Ben Mann, and Jared Kaplan. 2022. Training a helpful and harmless assistant with reinforcement learning from human feedback.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Erdem Bıyık, Malayandi Palan, Nicholas C Landolfi, Dylan P Losey, and Dorsa Sadigh. 2019. Asking easy questions: A user-friendly approach to active reward learning. *arXiv preprint arXiv:1910.04365*.

Erdem Biyik and Dorsa Sadigh. 2018. Batch active preference-based learning of reward functions. In *Conference on robot learning*, pages 519–528. PMLR.

Su Lin Blodgett, Solon Barocas, Hal Daumé III, and Hanna Wallach. 2020. Language (technology) is power: A critical survey of "bias" in NLP. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5454–5476, Online. Association for Computational Linguistics.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Wei Zhu, Yuan Ni, Guotong Xie, Zhiyuan Liu, and Maosong Sun. 2023. Ultrafeedback: Boosting language models with high-quality feedback.

Rudolf Flesch. 1979. How to write plain english. *University of Canterbury. Available at http://www. mang. canterbury. ac. nz/writing_guide/writing/flesch. shtml.[Retrieved 5 February 2016]*.

Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Vladimir I Levenshtein et al. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2022. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8086–8098, Dublin, Ireland. Association for Computational Linguistics.

William Muldrew, Peter Hayes, Mingtian Zhang, and David Barber. 2024. Active preference learning for large language models.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback.

Malayandi Palan, Nicholas C. Landolfi, Gleb Shevchuk, and Dorsa Sadigh. 2019. Learning reward functions by integrating human demonstrations and preferences.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36.

Zhizhou Ren, Anji Liu, Yitao Liang, Jian Peng, and Jianzhu Ma. 2022. Efficient meta reinforcement learning for preference-based fast adaptation. *Advances in Neural Information Processing Systems*, 35:15502–15515.

Michael J. Ryan, William Held, and Diyi Yang. 2024. Unintended impacts of llm alignment on global representation.

Dorsa Sadigh, Anca D Dragan, Shankar Sastry, and Sanjit A Seshia. 2017. *Active preference-based learning of reward functions*.

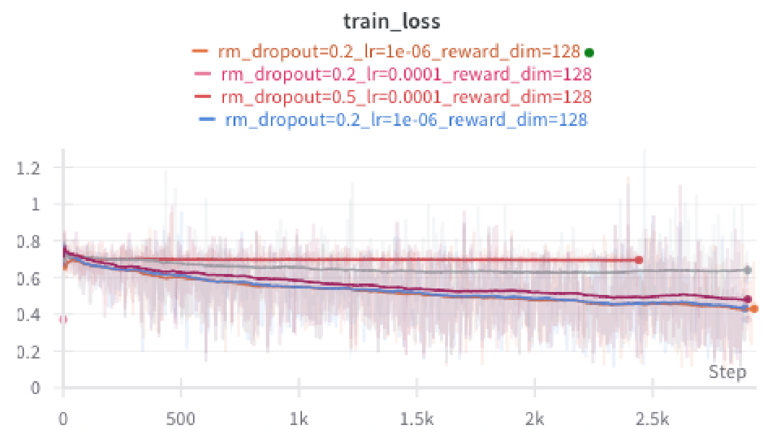

Figure 4: Training Loss over several thousand steps of training the Multi preference Reward Model with different hyperparameters.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Shengyi Huang, Kashif Rasul, Alexander M. Rush, and Thomas Wolf. 2023a. The alignment handbook. `https://github.com/huggingface/alignment-handbook`.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine Fourrier, Nathan Habib, Nathan Sarrazin, Omar Sanseviero, Alexander M. Rush, and Thomas Wolf. 2023b. Zephyr: Direct distillation of lm alignment.

Banghua Zhu, Evan Frick, Tianhao Wu, Hanlin Zhu, and Jiantao Jiao. 2023. Starling-7b: Improving llm helpfulness harmlessness with rlaif.

A Appendix

B Training Multi Reward Model

We provide a loss curve for the Multi Reward Model to show the learning over time of all eight training preferences in Figure 4

C Search over the preference space

Interestingly the search of both the length and lowercase preferences demonstrate similar divergence in lengths as shown in figure 5. This suggests that both may be doing similar searches over the preference space, but finding different weights to use for the preference embedding.

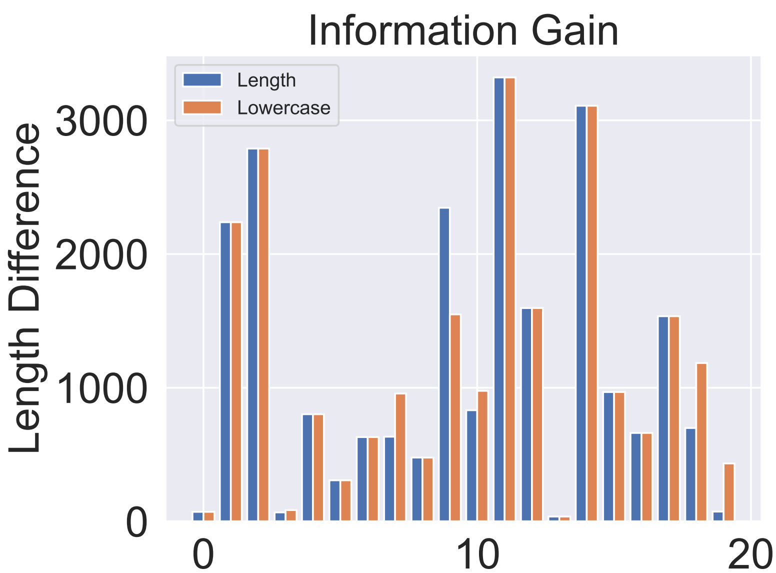

Figure 5: Comparison of the length of the selected comparisons by the Information Gain completion selector when actively learning both the length and the lowercase preference. The lengths are similarly divergent between both suggesting a similar search over the preference space in both optimizations. The X axis is query number and the Y axis the the difference in Lengths.