User-Centric and Parameter-Efficient Large Language Models

Guoyu Zhang

MInf Project (Part 1) Report
Master of Informatics
School of Informatics
University of Edinburgh

2024

Abstract

Large language models (LLMs) have revolutionized the field of natural language processing. Recent LLM-based chat assistants such as ChatGPT have exploded in popularity. Their effectiveness largely stems from techniques that align them to human preferences, such as Reinforcement Learning from Human Feedback (RLHF) and Direct Preference Optimization (DPO). However, these methods aim to align with the general population and fail to consider the diverse range of individual user preferences.

In this work, we present an exploration into aligning LLMs with user-specific preferences. We utilize Llama 2, DPO and Low-Rank Adaptation (LoRA) to efficiently capture user preferences with a small number of additional model parameters. There are three main components to our exploration: How effective can user-specific preferences be learned? Will merging adapters from different users improve generalization? How much data do we need? To this end, we also propose two new datasets: Stanford Human Preferences Subset (SHPS) and User-Specific Preferences (USP).

Our findings indicate that the ability of an LLM to capture preferences varies with the type of data. Although the LLM effectively captures preferences across both SHPS and USP datasets, its performance on SHPS is weaker, which we attribute to the dataset's more open-ended nature. Additionally, we discovered that merging LoRA adapters tuned on different users' data generally outperform the baseline and merging more adapters yields poorer results than merging just two. Finally, we determined that a minimum of 400 training samples is needed for meaningful improvements over the baseline. Though, the SHPS dataset continues to show a consistent increase in accuracy with additional samples, again attributed to its open-ended nature.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Guoyu Zhang*)

Acknowledgements

I am grateful to my supervisor Dr. Edoardo M. Ponti for his help and guidance, and Eddie for running many many hours of my experiments.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

In recent years, the field of natural language processing (NLP) has seen transformative changes with the emergence of large language models (LLMs) based on the Transformer architecture (Vaswani et al., 2017). Popular adaptations of LLMs for chat and dialogue-based applications include ChatGPT (OpenAI, 2024) and Llama 2 (Touvron et al., 2023a). The success of these models largely depends on their alignment with human preferences, achieved through fine-tuning the base LLM. Common techniques for alignment include Reinforcement Learning from Human Feedback (RLHF) (Ouyang et al., 2022) and Direct Preference Optimization (DPO) (Rafailov et al., 2024).

RLHF, DPO and recent research on alignment predominantly focus on aligning LLMs with general human preferences, or the preferences of specific user groups (Santurkar et al., 2023; Salemi et al., 2024). This ignores the diversity in user preferences and their different expectations from the LLM's responses (Casper et al., 2023). Additionally, user groups do not accurately reflect the opinions of individual users Kim and Lee (2024).

Sorensen et al. (2024) argue that artificial intelligence systems should reflect and support the diversity among humans. As the use of LLM-based chat assistants becomes more widespread, there emerges a greater necessity for these systems to specialize in order to support individual users effectively. However, fine-tuning an LLM for each individual user would be impractical due to their billion parameter sizes. Parameter-Efficient Fine-Tuning (PEFT) methods like Low-Rank Adaptation (LoRA) Hu et al. (2021) provide an effective solution by keeping the model weights frozen and introducing only a small number of trainable parameters.

We present an exploration into aligning LLMs with user-specific preferences by utilizing DPO and LoRA. This enables representation of user preferences through a small number of additional parameters. Furthermore, the additional parameters for each user are modular and can simply be switched out for training and inference, thus increasing efficiency and eliminating the need for multiple LLMs.

1.2 Objectives and Contributions

We aim to present an exploration into aligning an LLM, specifically Llama 2 (Touvron et al., 2023a), with user-specific preferences. Our work provides a basic foundation for efficient user-specific alignment for LLMs.

The DPO process uses binary options to fine-tune LLMs, intuitively guiding it to prefer one option over another. However, for our objectives, existing datasets for aligning LLMs to human preferences present a challenge. They typically comprise definitive right and wrong responses, effectively limiting user choices to a single option. Furthermore, these datasets aim to reflect general population preferences. Therefore, to capture user-specific preferences, we need a new dataset.

With this challenge in mind, our aims and contributions are as follows:

- **Create new datasets to capture preferences of individual users.**
  We introduce two new datasets. We filter the Stanford Human Preferences (SHP) (Ethayarajh et al., 2022) dataset and collect user annotations to compile the Stanford Human Preferences Subset (SHPS) containing preferences of 4 users on 1000 entries of human written, open-ended data. We modify the data generation procedure of Self-Instruct (Wang et al., 2023b) and collect user annotations to compile the User-Specific Preferences (USP) dataset which contains preferences of 4 users on 1000 entries of language model generated, distinct preferences data.

- **Evaluate and compare LLM effectiveness on capturing user-specific preferences for two different datasets.**
  We fine-tune models using SHPS and USP to show that the LLM effectively captures user preferences for both datasets. The LLM demonstrates a larger improvement in average accuracy from the baseline for the USP dataset with 17.5%, compared to a 7.75% for SHPS. This shows that open-ended preferences in SHPS are harder to capture for the LLM.

- **Evaluate and compare merging adapters fine-tuned on user-specific preference data for few-shot generalization for two different datasets.**
  We merge various combinations of adapters from different users to show an increase in accuracy above the baseline for an unseen user. Merging is more effective for SHPS, with an average accuracy increase of 5.08% from the baseline, compared to 1.92% for USP. This suggests adapters may overfit on USP's distinct preferences and fail to improve in generalization. Merging all adapters also perform worse than merging the two most aligned or two least aligned adapters.

- **Explore and compare quantity of data necessary for capturing user-specific preferences for two different datasets.**
  We use a varying number of training samples to determine the quantity needed to achieve meaningful improvements in accuracy. We show that 200 training samples is sufficient for minor improvement, but at least 400 samples is required for meaningful improvement. The USP dataset reaches the highest average accuracy at 400 samples, whereas the SHPS dataset continues to show a consistent increase in accuracy with additional samples, indicating that SHPS is more complex.

Chapter 2

Background

In this chapter, we provide an overview of the Transformer architecture, large language models (LLMs), and Parameter-Efficient Fine-Tuning (PEFT) techniques like Low-Rank Adaptation (LoRA). We also explore methods for aligning LLMs with human preferences, including Reinforcement Learning from Human Feedback (RLHF) and Direct Preference Optimization (DPO).

2.1 Transformers for NLP

2.1.1 Attention and Transformers

The transformer (Vaswani et al., 2017) is a deep learning architecture that excels in handling long range dependencies in sequences. Its defining feature is the self-attention mechanism. Though, the concept of attention in neural models for natural language processing (NLP) predates transformers. It was first introduced by Bahdanau et al. (2014) in encoder-decoder models. These encoder-decoder frameworks were initially applied to NLP by Cho et al. (2014) and Sutskever et al. (2014), and featured recurrent neural networks (Rumelhart et al., 1986). The RNN encoder takes a sequence of words as input, transform it into a single context vector, which is used by the RNN decoder to generate a sequence of outputs. With the addition of attention, the decoder is instead able to selectively mix information from a set of context vectors from the input, thereby allowing it to generate output focusing on the most relevant context from the input.

A limitation of RNN-based encoder-decoder models is that the RNN is sequential by nature, it maintains an internal state that is updated at each time step. The transform architecture introduced self-attention to propose a parallelized encoder-decoder model, which significantly increases efficiency. The self-attention mechanism utilizes query (Q), key (K), and value (V) matrices, each derived from the same input sequence but transformed through independently parameterized linear projections. The process starts by computing the dot product Q_iK_j of each query Q_i with each key K_j. This measures the relevance of every part of the input in relation to every other part. The dot products are scaled (by $\sqrt{d_k}$ – dimension of K) and normalized (by softmax) and can be used as weights. Recall that the third matrix V has been untouched, the weights are multiplied with corresponding values in V to determine how much each part of the input contributes to the output, or in other words, which parts of the input should be paid attention to. This is defined mathematically in Equation 2.1.

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$

Self-attention is used within both the encoder and decoder, while cross-attention bridges the gap between them. Cross-attention involve 2 sequences instead of 1. It takes K and V from the encoder and Q from the decoder, effectively connecting the input and output sequences. Building on self-attention, Vaswani et al. (2017) also introduce multi-head attention (MHA). Self-attention with a set of Q, K and V matrices can be considered as one head. MHA introduces multiple such heads to capture different features and relationships. Furthermore, due to the parallel nature of self-attention, information regarding positions for tokens/words in the sequence needs to be injected. Positional encodings are added to the embeddings of input sequences to address this.

2.1.2 Large Language Models

Large language models (LLMs) are built upon the Transformer architecture (Vaswani et al., 2017) and have revolutionized NLP. State-of-the-art LLMs such as GPT-4 (OpenAI, 2024) and Llama 2 (Touvron et al., 2023a) have demonstrated remarkable capabilities across various tasks including logical reasoning (Saparov et al., 2023), translation (Wang et al., 2023a), and code generation (Bubeck et al., 2023). These models are initially trained on massive datasets to gain a broad understanding of language. They can then be fine-tuned for specific tasks or domains through supervised fine-tuning (SFT) and aligned towards human preferences through Reinforcement Learning from Human Feedback (RLHF) (Ouyang et al., 2022).

In this project we use the Llama 2 family of LLMs, they are among the most widely-used open-source models. They are trained on a substantial corpus consisting of two trillion tokens of data and required up to 1.7 million GPU hours for the training process. At the time of release, they generally matched or surpassed the performance of other open-source and closed-source models (Touvron et al., 2023b). The Llama 2 family consists of pre-trained and fine-tuned LLMs, which are categorized into two types: Llama 2 and Llama 2-Chat. The key difference is that Llama 2-Chat models are fine-tuned for dialogue use cases. Both Llama 2 and Llama 2-Chat offer models with 7 billion, 13 billion, and 70 billion parameters.

2.2 Parameter-Efficient Fine-Tuning

While pre-trained language models (PLM) are equipped with extensive general language knowledge, they lack specialization in specific domains. Fine-tuning addresses this by training the PLM on domain-specific data, significantly improving its performance in downstream tasks. Traditionally, this is done through full fine-tuning, which involves adjusting all layers of the PLM. As language models grow in size, this becomes increasingly computationally expensive.

Parameter-Efficient Fine-Tuning (PEFT) techniques pose a solution. These approaches adjust only a small number of additional parameters, whilst keeping the original model's parameters frozen, and thus are modular in nature. Houlsby et al. (2019) first introduced the use of these small trainable modules in the context of NLP. Their approach attached small feedforward networks after each transformer sub-layer in the model architecture. These small modules are known as adapters.

Since then, various other PEFT methods have emerged, including prompt tuning (Lester et al., 2021) and prefix tuning (Li and Liang, 2021). These methods involve using soft prompts, which are trainable vectors attached to the embedding of the input to the language model. Specifically, prompt tuning adds a trainable vector only at the input, while prefix tuning introduces a vector at each transformer layer.

Hu et al. (2021) identify several limitations of these approaches. Adapters in Houlsby et al. (2019) introduce additional inference latency due to the extra computations required in those layers. Directly optimizing a prompt for prompt or prefix tuning can be challenging. Furthermore, allocating part of the sequence length for adaptation reduces the sequence length available for processing a downstream task. To address these issues, they propose Low-Rank Adaptation (LoRA), which we employ in this project. This approach is explored in detail in Section 2.2.1.

2.2.1 Low-Rank Adaptation

Low-Rank Adaptation (LoRA) (Hu et al., 2021) freezes the model's original weights and introduces trainable rank decomposition matrices into each layer of the transformer (Vaswani et al., 2017). This approach is based on findings by Aghajanyan et al. (2020) which show that common pre-trained models have a low intrinsic dimension, meaning it is possible to adjust only a small number of parameters to achieve almost the same performance as full fine-tuning. Hu et al. (2021) hypothesize that the weight update matrices also exhibit a low rank[1].

Given a model with weight matrix $W_0 \in \mathbb{R}^{d \times k}$ and its associated update matrix ΔW, LoRA decomposes ΔW into two smaller matrices $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$, where the rank $r \ll \min(d,k)$. During training, the weight matrix W_0 is frozen and only the decomposed weight update matrices A and B are changed. The weight update computation is as seen in Figure 2.1.

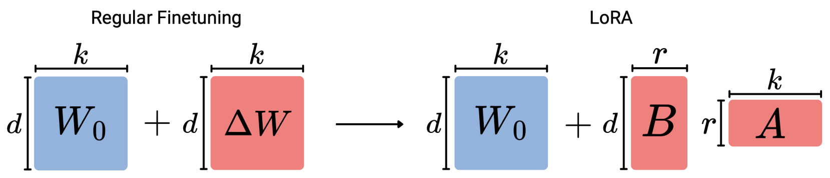

Figure 2.1: Figure in the same style as Miranda (2023). The weight update computation of regular fine-tuning and LoRA. Blue are frozen weights, red are weight update matrices.

[1]The rank of a matrix is the number of linearly independent rows or columns it has.

LoRA introduces two hyperparameters: the rank r, where a higher rank may improve performance at the cost of increased computational requirements, and the scaling factor α, which controls the degree of model adaptation to new training data. These hyperparameters specifically scale $\Delta W x$ in a forward pass by $\frac{\alpha}{r}$, where x is the input. The forward pass h is defined in Equation 2.2 as follows:

$$h = W_0 x + \Delta W x = W_0 x + BAx \tag{2.2}$$

Hu et al. (2021) apply LoRA specifically to the attention weights of the model, choosing not to extend it to other layers for simplicity. Later works by He et al. (2021) and Dettmers et al. (2023) found that applying LoRA to more layers enhance performance. Subsequent work build upon LoRA to introduce AdaLoRA (Zhang et al., 2023), which optimizes the allocation of trainable parameters across weight matrices and layers.

2.2.2 Adapter Merging

Fine-tuning a model for each downstream task can be a costly process. Recent model merging techniques present an approach to combine multiple models fine-tuned on task-specific data into one model. These techniques can also be applied to LoRA adapters. We will briefly describe one of these methods.

TIES (Trim, Elect, and Merge) Yadav et al. (2023) is an approach that can be used for merging LoRA adapters. It effectively deals with redundancy in parameters and disagreement in their update directions, which could lead to degraded performance in a merged model. There are three steps involved:

1. **Trim:** For all adapters, redundant parameters are “trimmed” by retaining only a set percentage of parameters with the highest magnitude and resetting the rest—those that have only changed marginally after fine-tuning—to zero.
2. **Elect**: An aggregated sign vector is created. This reflects the average direction of updates for each parameter across all adapters, where the average direction is either positive or negative.
3. **Merge:** For each parameter, the model averages the non-zero values across adapters that have the same sign as the sign in the aggregated sign vector.

2.3 Alignment Towards Human Preferences

Large language models (LLMs) are pre-trained on vast amounts of data, with objectives such as determining the contextual similarity between words and sentences (Devlin et al., 2019) or predicting the next word in a sentence (Radford et al., 2019). Although these processes enables LLMs to excel in various NLP tasks (Saparov et al., 2023; Bubeck et al., 2023), the training objectives do not specifically necessitate alignment with human preferences and values. Thus, in real world settings where prompts can be diverse, complex and malicious, these LLMs may produce undesirable and unsafe outputs (Gehman et al., 2020; Shah et al., 2019).

Reinforcement Learning from Human Feedback (RLHF) (Christiano et al., 2017; Ouyang et al., 2022) is a common method employed to ensure LLM outputs are aligned with human preferences. Inspired by the limitations in RLHF, Direct Preference Optimization (DPO) (Rafailov et al., 2024) presents an approach that deviates from reinforcement learning.

2.3.1 Reinforcement Learning from Human Feedback

Reinforcement Learning from Human Feedback (RLHF) (Christiano et al., 2017; Ouyang et al., 2022) employs reinforcement learning (RL) techniques to directly optimize a language model with human feedback. Typically, RLHF involves three key stages: 1) supervised fine-tuning (SFT), 2) reward modeling, and 3) fine-tuning with RL.

At the initial stage of RLHF, we begin with a pre-trained language model (LM). This model may undergo an optional SFT step which adapts it to specific downstream tasks. For the purposes of RLHF, this initial LM is kept frozen and acts as a reference to prevent the fine-tuned LM's probability distribution from diverging too much.

Next, we need to train a reward model. The aim is to obtain a model that is able to represent human preferences. When this model is given a sequence of text, it should be able to generate a scalar value indicating how well the text aligns with these preferences. To generate training data for this model, a common approach is to feed prompts to the initial LM to produce multiple responses for each prompt. These responses are then ranked by human annotators based on preference, and is then utilized to fine-tune or train another LM to produce the reward model.

Finally, with our initial LM and reward model, we can use RL to fine-tune a third LM which starts off as a copy of the inital LM. This model is referred to as the RL policy. In RL, the policy indicates actions that an agent takes as a function of its state and environment, in this case, it can be understood as an abstraction that describes the LM and its behaviour.

The three models together form the core components of RLHF. The RLHF process is as follows, with references to Figure 2.2 (note that the enumerations do not necessarily indicate strict order, they are only shown for convenience of illustration):

1. A prompt is given to the current iteration of the RL policy, and a response is generated.

2. The response is concatenated with the original prompt and fed into the initial model, which generates corresponding per-token probability distributions. These distributions are then compared with those from the RL policy, allowing us to calculate the deviations between the two models. The deviation is commonly measured with the KL divergence (Kullback and Leibler, 1951), which quantifies the difference between a probability distribution and a reference probability distribution.

   The KL divergence is used as a penalty in our reward function to ensure the RL policy does not shift too much from the initial LM. This ensures that the language modelling performance is maintained, as otherwise the LM could attempt to generate whatever is necessary to trick the reward model into giving favourable rewards, including generating nonsense.

3. The combined prompt with response from the RL policy is given to the reward model which assigns it a scalar score aiming to reflect human preferences.
4. The KL divergence penalty is summed with the scalar reward to obtain the final reward.
5. The final reward is sent to the RL update rule, which commonly uses proximal policy optimization (PPO) (Schulman et al., 2017). PPO is a RL method that aims to balance performance and stability of policy updates. PPO implements a clip function which constrains the policy update from being too large or too small. As too big of a step could cause the policy to deviate too much from the optimum whereas too small of a step is largely inefficient.
6. The update rule fine-tunes the RL policy for the current batch of data. This process then repeats for the next batch.

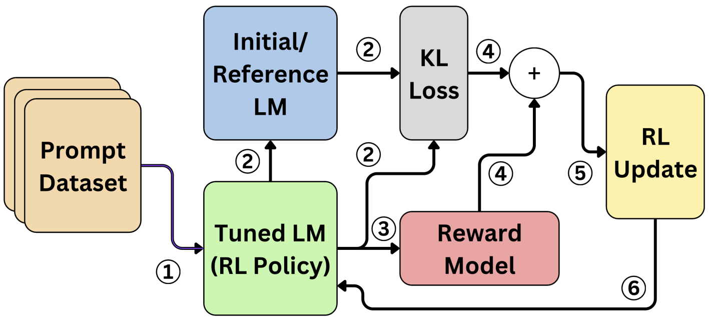

Figure 2.2: Overview of the RLHF process. Note that the numbers do not necessarily indicate strict order, they are only shown for convenience of illustration.

Mathematically, the objective that RLHF aims to optimize is defined in Equation 2.3. We want a high reward for the prompt and response from the reward model as represented by $r_\phi(x, y)$, and a low shift in probability distribution as represented by $\beta\mathbb{D}_{\text{KL}}\left[\pi_\theta(y \mid x) \mid\mid \pi_{\text{ref}}(y \mid x)\right]$.

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(y|x)}\left[r_\phi(x, y)\right] - \beta\mathbb{D}_{\text{KL}}\left[\pi_\theta(y \mid x) \mid\mid \pi_{\text{ref}}(y \mid x)\right] \tag{2.3}$$

Where x is a prompt in dataset $\mathcal{D}$, y is the response produced by the RL policy π_θ, r_ϕ is the reward model, π_{ref} is the initial LM which acts as a reference model, β is a parameter applied to the KL divergence $\mathbb{D}_{\text{KL}}$ controlling deviation from π_{ref}.

2.3.2 Direct Preference Optimization

Direct Preference Optimization (DPO) Rafailov et al. (2024) removes the need of a separate reward model which can be costly to train. However, a frozen reference model is still needed to ensure the distribution of the tuned model does not shift significantly. DPO uses a simple loss function, rather than the RL and PPO pipeline. It directly optimizes the model with preference data, which is a set of prompts along with a chosen and rejected response. The DPO process is as follows, with references to Figure 2.3:

1. For each data point, we combine the prompt with the chosen response and the prompt with the rejected response. These will be referred to as chosen and rejected respectively for the following discussion. Then, similarly to RLHF, we obtain the probability distributions for both chosen and rejected texts for the initial and tuned LMs.
2. The loss is calculated based on these probability distributions, with the aim of increasing the probabilities of chosen responses and decreases the probabilities of rejected responses. KL divergence Kullback and Leibler (1951) is again used to ensure minimal distribution shift from the reference model.
3. The tuned LM is optimized through backpropagation to minimize the loss. This then repeats for the next set of data.

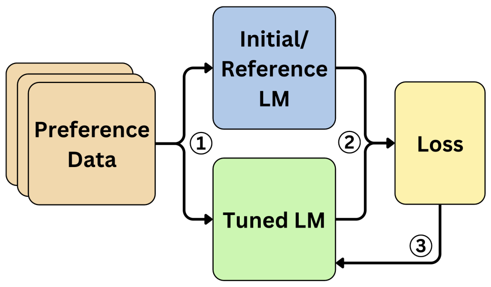

Figure 2.3: Overview of the DPO process.

Mathematically, DPO's loss function is defined in Equation 2.4. Intuitively, this aims to increase the difference between the winning responses $\frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)}$ and losing responses $\frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}$, while dividing by π_{ref} to ensure less distribution shift.

$$\mathcal{L}_{\text{DPO}}(\pi_\theta;\pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w\mid x)}{\pi_{\text{ref}}(y_w\mid x)} - \beta\log\frac{\pi_\theta(y_l\mid x)}{\pi_{\text{ref}}(y_l\mid x)}\right)\right] \tag{2.4}$$

Where x is the prompt, y_w is the winning/chosen response, y_l is the losing/rejected response, $\mathcal{D}$ is the dataset, π_θ is the LM being tuned, π_{ref} is the reference model, β is a parameter controlling deviation between π_θ and π_{ref}.

Since DPO eliminates the need for a reward model, it significantly enhances efficiency when adapting to new data. Furthermore, the nature of DPO's loss function enables models to be trained to both prefer and avoid specific topics. This is valuable as a large part of fine-tuning involves instructing the model to ignore certain subjects.

Chapter 3

Datasets

This chapter introduces the datasets employed in this project. We utilize and modify a well-known existing dataset, the Stanford Human Preferences dataset, and from it create what we refer to as the Stanford Human Preferences Subset (SHPS). Additionally, we propose a new dataset, which we will refer to as the User-Specific Preferences (USP) dataset.

3.1 Stanford Human Preferences Subset

The Stanford Human Preferences Subset (SHPS) is modified from the Stanford Human Preferences (SHP) (Ethayarajh et al., 2022) dataset. SHPS contains preference data for 4 users on 1000 distinct entries, each containing a human-written question and two corresponding responses.

We choose SHP because it offers a starting point for collecting user-specific preferences. Popular datasets used for aligning large language models to human preferences mostly contain definitive right and wrong responses. This includes datasets such as Anthropic's Helpfulness and Harmlessness (Bai et al., 2022), OpenAI's WebGPT (Nakano et al., 2022), and PKU's Safe-RLHF (Ji et al., 2023). On the other hand, SHP offers prompts with two responses that are both equally as valid.

The preference for a response is based on the collective preferences of Reddit[1] users. However, our goal is to align LLMs towards the preferences of individual users. Therefore, we modify this dataset and create a subset designed to capture the preferences of different individuals.

3.1.1 Original Dataset

The Stanford Human Preferences (SHP) (Ethayarajh et al., 2022) dataset is composed of approximately 385,000 entries. It is curated from the discussion forum Reddit and spans 18 diverse subject areas ranging from baking to anthropology. Each entry in the dataset

[1]Reddit is a social media website and discussion forum where content is created and voted on by site members.

is a Reddit post with a question or instruction and a pair of top-level comments[2] for the post. The preferred comment is determined by the number of votes each comment receives, aiming to reflect the helpfulness of one response over another based on the collective preferences of Reddit users.

Each entry in the dataset contains many metadata fields, such as the time of creation of comments, comment IDs, and post ID, among others. However, the core fields we are interested in are: **history**, which is the question or instruction; **human_ref_A**, the text of comment A; **human_ref_B**, the text of comment B; and **labels**, the preference label, which has either a value of 1 (A is preferred over B) or 2 (B is preferred over A). An example of these core fields is shown in Table 3.1

Field	Value
history	`Which research paper do you think was the funniest you've ever read? I'm just taking a day off of everything.`
human_ref_A	`The one where the biologist invents numerical integration.`
human_ref_B	`The case of the disappearing teaspoons: longitudinal cohort study of the displacement of teaspoons in an Australian research institute 10.1136/bmj.331.7531.1498`
labels	`1`

Table 3.1: An example entry from the SHP dataset, containing the following fields and their corresponding values: history, human_ref_A, human_ref_B, and labels.

3.1.2 Modifying the Dataset for Individual User preferences

3.1.2.1 Filtering the Original Dataset

We aim to modify the original dataset to reflect user-specific preferences. Given the original dataset's size of 385,000 entries, it is impractical and logistically impossible for users to review and annotate each one. Therefore, we need to reduce the dataset to a more reasonable number of samples. We aim to reduce the size down to 1000 high-quality entries. To accomplish this, we established specific filtering criteria, listed as follows:

- **Sequence Length:** We ensure that the combined length of the history with comment A and the history with comment B does not exceed a sequence length of 1024. This constraint ensures that the maximum sequence length of the language model we are working with is not exceeded.
- **Comment Score:** This is the number of upvotes[3] a certain comment received. We set the threshold so that both the scores for comment A and comment B are at

[2]A top-level comment directly responds to the original post, rather than replying to another comment.

[3]The number of upvotes is determined by the number of positive votes minus the number of negative votes plus 1, as defined by Reddit.

least 88[4], ensuring that the comments are both popular among users.

- **Score Ratio:** This is the ratio of the comment scores between comment A and comment B. We filter for a ratio less than or equal to 1.5 to ensure that the popularity of the comments is relatively equal.
- **URLs and Emojis:** We remove entries that contain URLs and emojis to help reduce noise within the data.
- **Duplicate Questions or Instructions:** Some entries in the dataset relate to the same post, it contains the same question but presents different comments within the post for comment A and comment B. We keep only one entry relating to the same post to reduce noise in the data and cover a wider range of domains.

After filtering, we obtained 1,012 entries and selected the first 1,000 for our modified subset. We then shuffled the order to help ensure that the distribution of domains of the text is uniform across the entire dataset.

3.1.2.2 Collecting Individual User Preferences

The collection process involves a simple Python script that cycles through each dataset entry and presents it to the user through the command-line interface. The user is shown the **history** (instruction or question), **human_ref_A** (comment A) and **human_ref_B** (comment B). They are then asked to choose between 1 or 2, where 1 indicates a preference for **human_ref_A** and 2 indicates a preference for **human_ref_B**. The chosen preference, 1 or 2, is stored in the corresponding **labels** field. This process is time-intensive, requiring about 6 hours per user to complete. We conducted this process with four anonymous users.

3.1.2.3 Compiled Dataset

After the collection process, we obtain preferences from four different users based on the same 1000 entries of the filtered SHP data. For each entry, these preferences are combined into one **labels** field, formatted as a list, where the index of the list represents the user number minus one, due to zero-based indexing.

We perform basic analysis to investigate the correlation between users' preferences, using a method inspired by the Hamming distance (Hamming, 1986). The Hamming distance between two strings or vectors of equal length measures the number of positions at which the corresponding symbols differ. However, in our analysis, we adapt this approach to identify the number of positions where the symbols—or in our case, preferences—are the same, thus indicating agreement between users.

We calculate the agreement for each pair of users across every 50 entries. We also aggregate these results across all 1000 entries to provide an overall view of user agreement. These results are presented in Figure 3.1.

[4]We want to obtain 1000 entries of data, this is the highest threshold that retains just over 1000 entries, given our other filtering criteria.

Correlations between users' preferences are indicated visually by the color intensity in Figure 3.1. Specifically, darker colors suggest strong positive agreement (and thus positive correlation) between user pairs, and light colors indicate a strong negative correlation.

We observe that the colors representing agreement for segments between each pair of users are mostly neutral, neither dark nor light, suggesting that there is not a lot of correlation among users. This observation is further supported by the total agreement scores for pairs of users, which span a range of 461 to 520. The values are close to the 500 midpoint, which represents a 50% agreement rate and indicates a lack of correlation, as statistically, random assignments in sufficiently large binary data of two users would be expected to converge on 50% agreement[5].

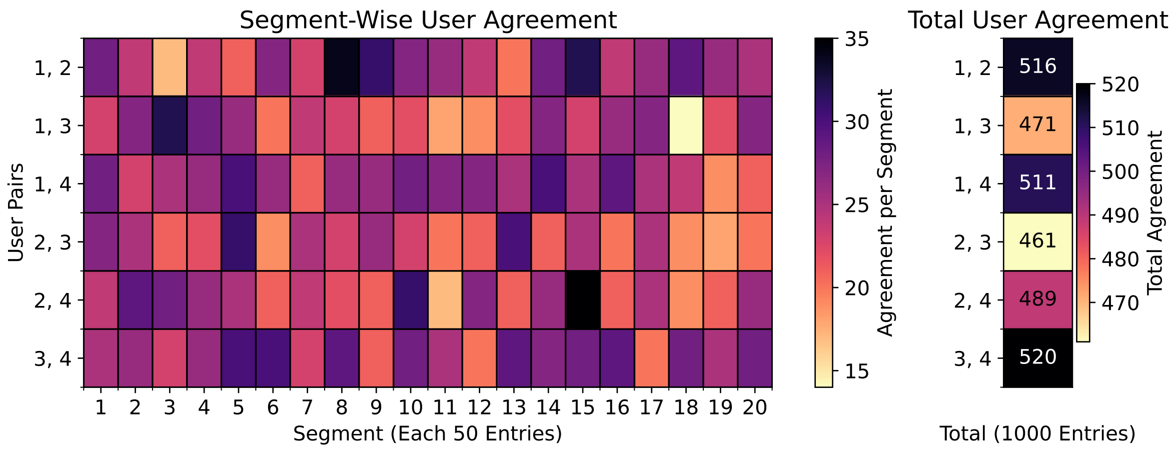

Figure 3.1: The agreement between user preferences for the SHPS dataset. Left shows agreement broken down into segments of 50 entries each. Right shows total agreement in 1000 entries. Where agreement is the number of entries where users share the same preference.

3.2 User-Specific Preferences

We propose the User-Specific Preferences (USP) dataset as a novel dataset designed to capture preferences of individual users. The data is generated by a large language model (LLM) and annotated by 4 users. Similar to the Stanford Human Preferences Subset (SHPS), USP also comprises 1000 entries, each containing unique instructions with 2 corresponding responses.

As mentioned in Section 3.1, current datasets for aligning LLMs with human preferences typically contain instructions with a right and wrong response. For example, in Anthropic's Helpfulness and Harmlessness dataset (Bai et al., 2022), responses are

[5]This is expected due to the possible outcomes of choices between two users. There are two cases where users can agree: both selecting comment A or comment B. There are also two cases for disagreement: one user preferring comment A and the other preferring comment B, and vice versa. The probabilities for agreement and disagreement is therefore both 50%.

categorized into helpful versus unhelpful, or harmful versus harmless. Additionally, these datasets aim to reflect the preferences of the general human population rather than individuals.

The SHPS dataset proposes an approach to address these factors. However, upon further review of this dataset, we notice that selecting a preferred option between comments can sometimes seem arbitrary. This is because the comments are mostly open-ended and do not exhibit a clear theme or domain. An example in Table 3.1 illustrates this, as there does not seem to be a clear reason to pick one response over the other. To improve on this, we propose the USP dataset, which comprises responses that represent distinct user preferences.

Table 3.2 provides an example of an entry from the USP dataset. It includes several fields: **instruction**, **response_1**, **response_2**, and a **chosen** field that indicates the user's preference. A value of 1 in **chosen** means that the user prefers **response_1**, while a value of 2 means a preference for **response_2**. We can see that the themes or domains of the responses are obvious; we can generalize that the main difference is that **response_1** is about indoor activities, while **response_2** is about outdoor activities.

Field	Value
instruction	`I'm trying to be more physically active. What are some fun and unique exercises I can try?`
response_1	`Dance your way to fitness with upbeat and fun exercise classes like Zumba, Hip Hop, or Bollywood. These activities can be a great combination of cardio and fun.`
response_2	`Take advantage of the outdoors by trying activities like hiking, rock climbing, or surfing. These can provide a full-body workout while enjoying the beauty of nature.`
chosen	`1`

Table 3.2: An example entry from the USP dataset, containing the following fields and their corresponding values: instruction, response_1, response_2, and chosen.

3.2.1 Data Generation

Generating large amounts of instruction and response data manually can be challenging, as it requires not only creativity to come up with diverse instructions and responses but also because of the sheer volume involved. To address this, we use a large language model (LLM) for data generation, adapting the data generation procedure of the Self-Instruct framework (Wang et al., 2023b). This allows us to efficiently produce a wide range of data for gathering user preferences.

The Self-Instruct framework aims to fine-tune a language model based on its own generations. An overview is as follows, the pipeline starts by initiating a task pool from a seed set of tasks. Random tasks from the task pool are sampled and combined into the prompt template to prompt an LLM to generate new instructions along with its associated inputs and outputs. The generated tasks are then filtered to remove invalid and highly similar samples before being added to the task pool. This process repeats until the desired amount of tasks are generated. The tasks are then used to fine-tune the LLM itself.

For our purposes, we are only interested in the data generation procedure of this framework. To align the data generation procedure with our objectives, we make **only** the following changes to the original process and code of Wang et al. (2023b):

- **LLM for Data Generation:** Wang et al. (2023b) uses the Davinci engine of the GPT-3 (Brown et al., 2020) language model for data generation. However, as this engine is no longer available at the time of our project, we instead opt for GPT-3.5 Turbo Instruct[6]. This newer model builds upon the original GPT-3 and subsequent InstructGPT series (Ouyang et al., 2022) models for improved instruction-following capabilities.

- **Seed Tasks:** The original Self-Instruct framework utilized 175 diverse seed tasks, including classification questions, math questions, and instructions asking the model to perform specific tasks. However, these types of instructions are not desirable for our objectives, therefore we create new seed tasks.

  Given the challenges associated with manually generating a large amount of instruction and response data, we choose to create a smaller number of 30 seed tasks. We manually select 30 instructions and their corresponding two responses to reflect distinct preferences.

  These instructions and responses are generated using GPT-4 Turbo[7], an improved version of GPT-4 (OpenAI, 2024) which achieves human-level performance on many tasks and at the time of our project, leads the LMSys Leaderboard (Chiang et al., 2024) designed for evaluating LLMs. We choose to use GPT-4 Turbo for only seed task generation and not all data generation due to its higher cost compared to GPT-3.5 Turbo Instruct[8]. The generation is completed through OpenAI's Assistants platform[9]. The seed tasks compiled are available in Appendix A.1.

- **Prompt Template:** The Self-Instruct framework uses a basic prompt template that simply requests the model to generate tasks. Later work by Taori et al. (2023) in Stanford's Alpaca project adapts and modifies this framework to generate data for fine-tuning a 7 billion parameter Llama (Touvron et al., 2023a) model. One modification Alpaca makes is that it creates a new prompt template which provides explicit task generation requirements to the model. We adopt Alpaca's revised prompt template and make further modifications. Most importantly, we add a specific requirement specifying that the model must generate an instruction

[6]`https://platform.openai.com/docs/models/gpt-3-5-turbo`

[7]`https://platform.openai.com/docs/models/gpt-4-turbo-and-gpt-4`

[8]GPT-3.5 Turbo Instruct costs \$1.50 / 1M tokens for input and \$2.00 / 1M tokens for output, whereas GPT-4 Turbo costs \$10.00 / 1M tokens for input and \$30.00 / 1M tokens for output, as listed at `https://openai.com/pricing`

[9]`https://platform.openai.com/assistants/`

accompanied by two responses, labelled as A and B, reflecting the preferences of two opposite users. All modifications we make to Alpaca's template can be found in Appendix A.2.

- **Instance Generation:** As mentioned previously, the Self-Instruct framework generates various types of instructions, including classification questions, which are undesirable for our purposes. Such questions often require an input along with the instruction and output. Wang et al. (2023b) uses what they call instance generation to help the model figure out what inputs are needed and generate them. Since we do not require inputs for the tasks we generate, the instance generation step is omitted from our procedure.

- **Similarity Metric:** To ensure diversity is maintained among generated tasks, Wang et al. (2023b) only adds a new instruction to the task pool if its ROUGE-L[10] (Lin and Och, 2004) similarity with any existing instruction is less than 0.7.

  ROUGE-L is able to measure syntactic similarities between two texts. However, it does not necessarily capture semantic similarities as it focuses solely on the order and presence of exact words, disregarding the meanings behind them. Therefore, during an initial exploration of the data generation procedure, we find a large amount of syntactically dissimilar but semantically similar tasks.

  To address these limitations, we aim to measure similarity of instructions through the cosine similarity[11] of their word embeddings. We propose using BERT (Devlin et al., 2019) to obtain the embeddings, due to its state-of-the-art performance on the Semantic Textual Similarity benchmark (Cer et al., 2017). We utilize the **SentenceTransformers**[12] (Reimers and Gurevych, 2019) library to compute the cosine similarity between embeddings of each generated instruction and all other instructions in our task pool, keeping only instructions lower than a 0.9 threshold. This threshold is chosen as lower thresholds significantly increase generation time[13] while higher thresholds produce extremely similar instructions.

With the modified procedure, as illustrated in Figure 3.2, we generate over 1000 instructions each with two responses. Then the dataset is manually reviewed and filtered down to 1000 entries, removing invalid instructions such as those asking for an AI assistant to perform tasks or asking general knowledge questions. The total cost for obtaining 30 seed tasks and 1000 data entries is $15, with $13 allocated for experimentation and $2 for generation[14].

[10]ROUGE-L measures text similarity based on the longest common subsequence (LCS) between two text sequences. It identifies the longest series of words that appear in the same order in both texts, though not necessarily consecutively. A longer LCS indicates a greater similarity, as it reflects a significant overlap in content.

[11]Cosine similarity for word embeddings is commonly used in natural language processing Mohammad and Hirst (2012), it evaluates the cosine of the angle between two vectors as a measure for similarity.

[12]Version 2.7.0

[13]The time taken to generate 1000 tasks with a threshold of 0.9 is around 10 minutes, while the time taken to generate 1000 tasks with a threshold of 0.7 is over 5 hours.

[14]For detailed pricing, see: `https://openai.com/pricing`

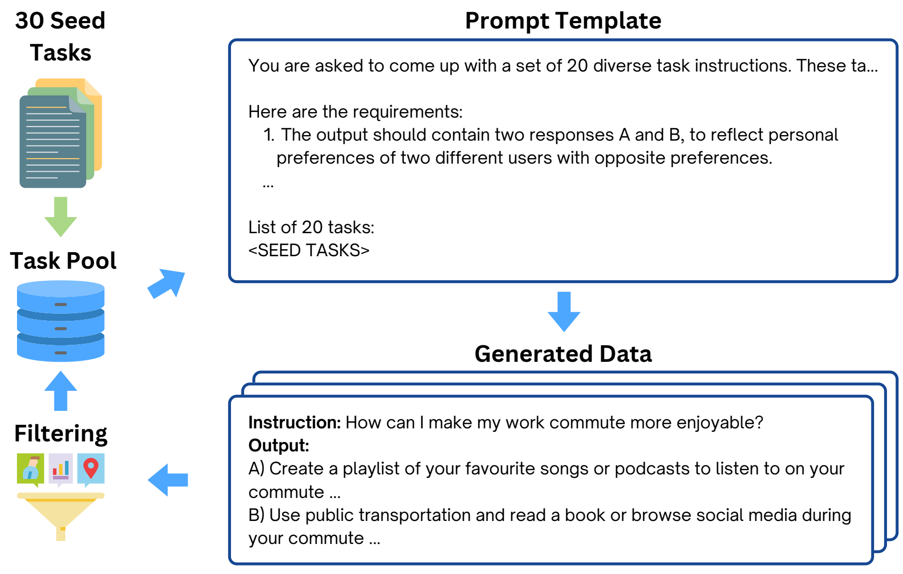
30 Seed Tasks
Task Pool
Filtering
Prompt Template
You are asked to come up with a set of 20 diverse task instructions. These ta...
Here are the requirements:
1. The output should contain two responses A and B, to reflect personal preferences of two different users with opposite preferences.
...
List of 20 tasks:
<SEED TASKS>
Generated Data
Instruction: How can I make my work commute more enjoyable?
Output:
A) Create a playlist of your favourite songs or podcasts to listen to on your commute ...
B) Use public transportation and read a book or browse social media during your commute ...

Figure 3.2: An overview of our modified data generation procedure. 30 seed tasks are added to the task pool. Tasks are sampled from the pool and combined with the prompt template. The LLM is prompted to generate task data, which is filtered before being added to the task pool. The green arrow indicates a process that happens once, while the blue arrow shows processes that repeats until the desired quantity of tasks is generated.

3.2.2 Analysis of Generated Data

We present a basic overview on the diversity of the generated data, following the methodology in the Self-Instruct paper (Wang et al., 2023b). The Berkeley Neural Parser[15] (Kitaev and Klein, 2018; Kitaev et al., 2019) is utilized to parse the instructions and extract the root verb and its first direct noun object[16]. The top twenty most common root verbs and their top 4 direct noun objects are presented in Figure 3.3. We analyze the length and similarity of generated instructions and responses and plot the distribution in Figure 3.4.

Despite Figure 3.3 showing a wide range of instructions, Figure 3.4 reveals that the distribution of similarity scores is left-skewed, indicating that many instructions are quite similar to each other. This is not ideal as we aim to obtain distinct instructions to capture diverse user preferences. However, this stems from the inherent limitations in creativity associated with LLMs. Due to their auto-regressive nature, they struggle to generate new and suprising products (Bunescu and Uduehi, 2019). Furthermore, these models tend to replicate the data distributions they were trained on (Shanahan, 2023). Future work could explore methods of incorporating creativity within the data generation procedure.

We observe reasonable diversity in the lengths of instructions and responses. Interest-

[15]`https://parser.kitaev.io`

[16]A direct object is a noun or noun phrase that receives the action of a verb in a sentence.

ingly, the distribution of the length of response 2 is shifted to the right compared to that of response 1. The implications of this shift are not immediately clear and could be explored in future work.

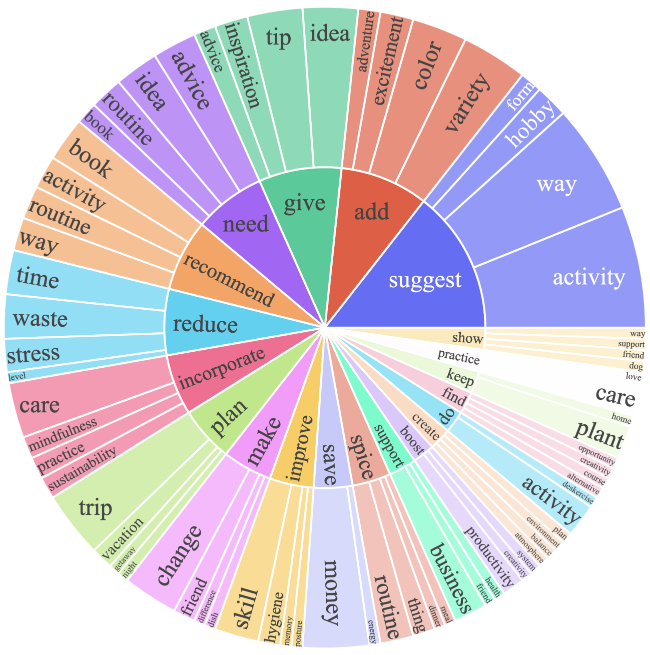

Figure 3.3: The top 20 most common root verbs (inner circle) and their top 4 direct noun objects (outer circle) in the generated instructions.

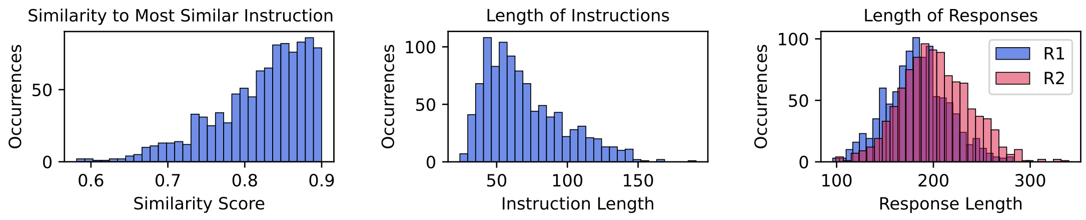

Figure 3.4: Distributions of instructions and responses for similarity and length. Left: Similarity between generated instruction and its most similar instruction. Middle: Length of instructions. Right: Length of responses to instructions, where R1 is response 1 and R2 is response 2.

3.2.3 Collecting Individual User Preferences

The collection process is the same as for the Stanford Human Preferences Subset (SHPS). It involves a simple Python script that cycles through each dataset entry and presents it to the user through the command-line interface. The user is shown the **instruction**, **response_1** and **response_2**, and is asked to choose between 1 or 2. The chosen preference is stored in the **chosen** field. This process is time-intensive, requiring about 5 hours per user to complete. We conducted this process with four anonymous users.

3.2.4 Compiled Dataset

As with our compiled Stanford Human Preferences Subset (SHPS) dataset, after the collection process, we also obtain preferences from 4 different users based on the same 1000 entries of our generated data. For each entry, these preferences are combined into one **chosen** field, formatted as a list, where the index of the list represents the user number minus one, due to zero-based indexing.

Figure 3.5 is generated following the same approach as Section 3.1.2.3. Also, as previously discussed, correlations between users' preferences in the heatmap are visualized through color intensity: dark colors signify positive correlation, while light colors denote negative correlation.

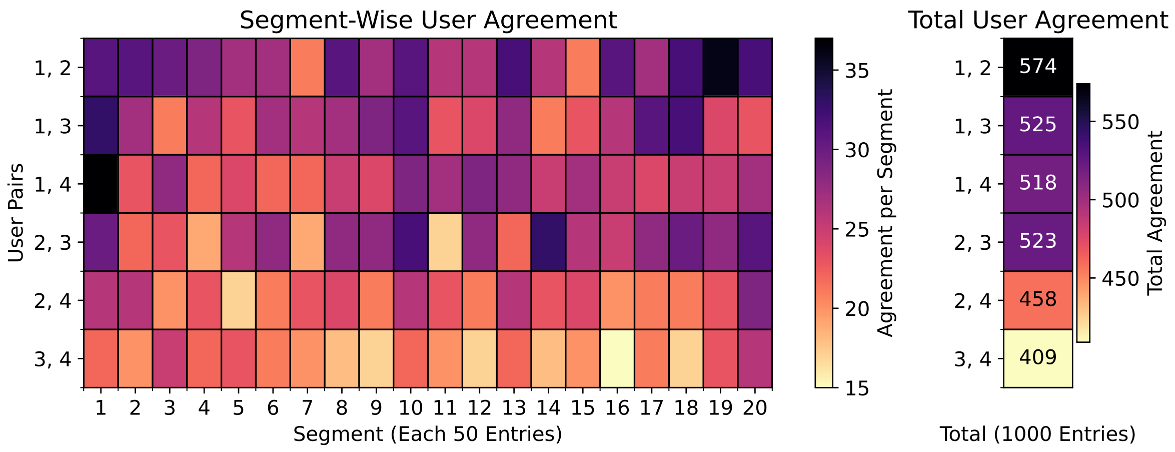

Figure 3.5: The agreement between user preferences for the USP dataset. Left shows agreement broken down into segments of 50 entries each. Right shows total agreement in 1000 entries, where agreement is the number of entries where users share the same preference.

We observe that the colors representing agreement for segments between each pair of users are again generally neutral, neither dark nor light, suggesting a lack of correlation. However, there are two pairs of users that stand out. Firstly, for the pair consisting of user 1 and 2, we observe many segments have a dark color, suggesting a positive correlation. This is further supported by the fact that they preferred the same response in 574 out of 1000 entries. On the other hand, for the pair consisting of user 3 and 4, many segments have light colors, indicating a negative correlation between them. This observation is supported by the fact that they only prefer the same response in 409 out of 1000 entries.

SHPS had total agreement scores spanning a range of 461 to 520, whereas the User-Specific Preferences (USP) data shows a larger deviation from the 50% midpoint, with total agreement scores spanning a broader range of 419 to 574. This shows there are stronger correlations among users, both positive and negative. This is likely because of the design of the USP dataset, which comprises responses representing more distinct preferences, rather than more open-ended comments in the SHPS data.

3.3 Summary

This chapter outlined the two datasets employed in this project, the process of obtaining them and some basic analysis is summarized as follows:

- We utilize the **Stanford Human Preferences Subset (SHPS)** and **User-Specific Preferences (USP)** datasets.
- Each dataset contains **1000 entries**, where each entry includes an instruction, two responses, and a preference showing which response is favored. We collected preferences from **4 users** for each dataset.
- The **SHPS** dataset is initially obtained by filtering the Stanford Human Preferences (SHP) dataset (Ethayarajh et al., 2022). It consists of **human-written data** sourced from Reddit.
- Initial analysis of the **SHPS** dataset reveals that the data is **diverse and open-ended**, with **little correlation** between user preferences.
- The **USP** dataset comprises **language model-generated data**. We modified the data generation procedure from Self-Instruct (Wang et al., 2023b) to create this dataset.
- The **USP** dataset is designed to capture **distinct preferences**, and initial analysis indicates **moderate diversity**, with some instructions showing high similarity. Further analysis reveals a **higher correlation** among user compared to the **SHPS** dataset.

Chapter 4

Methodology

This chapter establishes the main components for conducting experiments outlined in the subsequent Experiments chapter. Our main goal is to fine-tune a baseline model on user-specific data and explore adapter merging for generalization. To achieve this, we introduce the baseline model, detail data preprocessing steps, and outline fine-tuning and adapter merging implementations. We also introduce methods used for evaluation and note the compute environment and runtimes for experiments.

4.1 Baseline

In this project, we selected the Llama 2-Chat 7B model from the Llama 2 family (Touvron et al., 2023b) as the baseline. This model is selected for two reasons. First, it is the smallest model in the series, with 7 billion parameters, which fits within our computational budget compared to its larger counterparts with 13 billion and 70 billion parameters. Secondly, it is fine-tuned for dialogue use cases. Given that our data—Stanford Human Preferences Subset (SHPS) and User-Specific Preferences (USP)—consists of dialogue-style interactions with instructions and responses, this removes the need for supervised fine-tuning to a dialogue task before Direct Preference Optimization (DPO).

For most of the implementation, we use open-source libraries from Hugging Face[1]. To obtain our baseline Llama 2-Chat 7B[2] model we make use of the **Transformers**[3] (Wolf et al., 2019) library. **Transformers** is based on the machine learning library PyTorch[4] (Ansel et al., 2024) and provides APIs to download and use thousands of pretrained models. These models can be used for direct inference or further fine-tuning. Each model architecture within **Transformers** is also fully standalone, and can be easily modified.

[1]`https://huggingface.co`

[2]Available at: `https://huggingface.co/meta-llama/Llama-2-7b-chat-hf`

[3]See Appendix B.1 for version number.

[4]See footnote 7

4.2 Data Preprocessing

The Llama 2-Chat models employ a specific prompt template in the training procedure (Touvron et al., 2023b), as shown in Table 4.1. During fine-tuning, it is important that the format of our data fits this template so that the expected features and performance can be achieved. The template uses specific markers to delineate different parts of the message to the model: `<s>` and `</s>` indicate the beginning and end of a full conversation, which includes the user's message and the model's response; `[INST]` and `[/INST]` encapsulate user messages; and `<<SYS>>` and `<</SYS>>` define the boundaries of the system prompt. The latter provides essential context that guides the model's responses. To illustrate, the system prompt set by Touvron et al. (2023b) begins with "You are a helpful, respectful and honest assistant...". This template supports multi-turn conversations by appending each new user message and response to the ongoing dialogue thread.

Our datasets SHPS and USP each comprise only a single user message accompanied by two responses to this message within each entry. Thus, in transforming the data to the required format, we are focusing on single-turn conversations. In this process, we have chosen not to include a system prompt for simplicity. For the user prompt or message, we enclose these within `<s>[INST]` at the beginning and `[/INST]` at the end. For both the chosen and rejected responses, we append `</s>` at the end of the response to ensure that when a prompt is combined with a chosen or rejected response, the correct format is maintained. An example of a USP prompt combined with a chosen response is shown in Table 4.1.

Llama 2-Chat Template

```
<s>[INST] <<SYS>>
{{ system_prompt }}
<</SYS>>

{{ user_msg_1 }} [/INST] {{ model_answer_1 }} </s>
<s>[INST] {{ user_msg_2 }} [/INST]
```

Processed Data

```
<s>[INST] Suggest something I can do to relax after a long day.
[/INST] Spend quality time with loved ones by having a game night,
going out for dinner, or watching a movie together.  Socializing
can be a great way to de-stress and reconnect with others.  </s>
```

Table 4.1: The Llama 2-Chat template and an example of processed data from the USP dataset to fit this format, where the prompt is in blue and chosen response in green.

Both the SHPS and USP datasets are processed as described, utilizing the Hugging Face **Datasets**[5] (Lhoest et al., 2021) library. This library leverages the Apache Arrow (Richardson et al., 2024) format to process large datasets efficiently and also features in-

[5]See Appendix B.1 for version number.

tegration with the Hugging Face Hub[6] which allows datasets to be easily shared. The processed datasets are available at `https://huggingface.co/datasets/guoyu-zhang`.

4.3 Fine-Tuning

We utilize the Hugging Face **PEFT**[7] (Mangrulkar et al., 2022) library for training Low-Rank Adaptation (LoRA) (Hu et al., 2021) adapters for each user. The Parameter-Efficient Fine-Tuning (PEFT) library implements many state-of-the-art PEFT methods, and is integrated with the **Transformers** library. This sets up the basic framework required to fine-tune the Llama 2-Chat 7B model.

The Hugging Face **TRL**[8] (Werra et al., 2020) library is utilized for the Direct Preference Optimization process (DPO) (Rafailov et al., 2024). **TRL** is a library which provides a set of tools to train transformer-based language models with reinforcement learning, it is also integrated with the **Transformers** library. This allows DPO training on LoRA adapters of the Llama 2-Chat 7B model for each user's data.

The LoRA adapter for each user, trained on their respective data from the SHPS and USP datasets, is available at: `https://huggingface.co/guoyu-zhang`.

4.4 Adapter Merging

To merge the trained adapters for inference we again utilize the Hugging Face **PEFT** (Mangrulkar et al., 2022) library. **PEFT** provides several methods for model merging, we opt for an efficient method for merging LoRA adapters: Trim, Elect, and Merge (TIES) (Yadav et al., 2023). TIES effectively deals with redundant and sign disagreement in parameters which could lead to degraded performance in a merged model, as explained in Section 2.2.2.

4.5 Evaluation

When evaluating the performance of models fine-tuned on a specific user's data, we load the specific adapter associated with that user onto the baseline model and conduct evaluation using their SHPS and USP test sets. The process for evaluating merged adapters follows the same approach, we also load the merged adapter onto the baseline model and perform evaluation on the chosen test set. Detailed explanations for the setup and how the test set is chosen can be found in the experiment setup section of each experiment (see Sections 5.1.2, 5.2.2, and 5.3.2).

Since there is no standardized benchmark for evaluating how well a model has been fine-tuned to individual user preferences captured in our proposed datasets, we use well-established evaluation metrics from the fields of machine learning and natural language processing. Specifically, we measure accuracy to gauge overall model performance and

[6]`https://huggingface.co/docs/hub/en/index`

[7]See Appendix B.1 for version number.

[8]See footnote 7.

perplexity to assess how well the model predicts the test data and also to gauge the shift in the model's probability distribution.

4.5.1 Accuracy

Accuracy is the ratio of accurately predicted observations to the overall number of predictions made. This metric is formally defined in Equation 4.1 as follows:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{4.1}$$

In this project, we measure accuracy by examining how likely the model is to generate specific responses to given prompts. Our goal is to determine whether the model is more inclined to produce a chosen response rather than a rejected one. For a particular sample, a prediction is deemed correct when the average negative log-likelihood (NLL), as defined in 4.2, of the model generating the prompt combined with the chosen response is lower than that for the prompt combined with the rejected response. We use average negative log-likelihood because it quantifies the likelihood of each response effectively; a lower value indicates a greater probability that the model will select the response given the prompt. This approach provides a basis for comparison of the baseline, fine-tuned models/adapters and merged adapters.

$$-\overline{\mathcal{L}}(W) = -\frac{1}{N}\sum_{i=1}^{N} \log P(w_i|w_1,...,w_{i-1}) \tag{4.2}$$

where $W = w_1, w_2, ..., w_N$ represents the sequence of words, $P(w_i|w_1,...,w_{i-1})$ is the probability that the language model assigns to the word w_i given all the preceding words $w_1,...,w_{i-1}$, and N is the total number of words in the sequence.

4.5.2 Perplexity

Perplexity is first introduced in Jelinek et al. (1977) for speech recognition. It is a measure of uncertainty, quantifying how well a probabilistic model predicts a sample. A lower perplexity indicates that the model is more certain about its predictions, whereas a higher perplexity suggests the model is less confident. Mathematically, perplexity is the exponential of the entropy or, equivalently, the average negative log-likelihood of a given sequence, and is defined as follows in Equation 4.3.

$$\text{Perplexity}(W) = \exp\left(-\overline{\mathcal{L}}(W)\right) \tag{4.3}$$

where $-\overline{\mathcal{L}}(W)$ represents the average negative log-likelihood of the sequence $W = w_1, w_2, \ldots, w_N$ as defined in Equation 4.2, and exp is the exponential function.

In our case, perplexity is not only used to assess the model's generalization capabilities over unseen data but also to ensure the Direct Preference Optimization (DPO) process does not significantly alter the model's probability distribution. Maintaining this distribution is important, as significant alterations could result in the model generating unexpected outputs, thereby degrading its overall performance. The perplexities we report in our experiments are all for the prompt combined with the chosen response, as we are mainly interested in how well the model is aligned to chosen responses.

4.6 Compute Environment and Runtimes

The experiments, which includes fine-tuning the baseline model with LoRA on the SHPS and USP datasets, as well as merging adapters and evaluating these methods, were conducted on Eddie (ECDF), the University of Edinburgh's research compute cluster. Each experiment makes use of a single NVIDIA A100 GPU with 80GB of GPU RAM.

The runtimes for the experiments are detailed as follows: fine-tuning for all experiments on the SHPS dataset took 1.5 hours, while for the USP dataset, the experiments took 1 hour. The shorter duration for the USP dataset may be attributed to its data features being more easily captured during the training process.

4.7 Summary

This chapter covered several core components for our research implementation:

- In the implementation of our methods, we primarily employed open-source Hugging Face libraries: **Transformers**, **Datasets**, **PEFT**, and **TRL**.
- The **Llama 2-Chat 7B** model was selected as the baseline for further fine-tuning. This model requires a specific input template so we **preprocess SHPS and USP datasets to conform to the template**.
- We evaluate our baseline, fine-tuned models/adapters, and merged adapters using **accuracy** and **perplexity** as key metrics to assess their performance.
- Our experiments were conducted on a single **NVIDIA A100 GPU with 80GB of GPU RAM**, with each experiment requiring approximately 1.5 hours for the SHPS dataset and 1 hour for the USP dataset.

Chapter 5

Experiments

This chapter describes the experiments conducted to examine how well a large language model (LLM), specifically Llama 2-Chat 7B, aligns with user-specific preferences across two datasets. We conduct three experiments: firstly, evaluating and comparing the LLM's effectiveness in capturing user-specific preferences; secondly, assessing and comparing the performance of merging adapters that have been fine-tuned on user-specific preference data for few-shot generalization; and thirdly, exploring and comparing the amount of data required to effectively capture user-specific preferences. **Note: For the remainder of this chapter, we directly refer to the user's "adapter" and "fine-tuned adapter" for discussion, though in practice the adapter is loaded onto the baseline model then used for inference.**

5.1 Learning User-Specific Preferences

5.1.1 Objective

Previous research has predominantly aimed at aligning models with general human preferences. This experiment aims to investigate whether a model can be fine-tuned to align with the preferences of specific individuals and to what extent. To this end, we utilize the Stanford Human Preferences Subset (SHPS) and User-Specific Preferences (USP) datasets. SHPS is derived from Reddit and is human-written, it also reflects a broad range of preferences. In contrast, USP is language model-generated and designed to represent two distinct preferences. This experiment will evaluate how effectively the model can align with these individual preferences and examine the performance differences between the two datasets.

5.1.2 Experimental Setup

We begin by fine-tuning the baseline model on both the SHPS and USP datasets. Separate adapters are fine-tuned for each user within each dataset, resulting in a total of 8 distinct adapters. As detailed in Section 4.2, both the SHPS and USP datasets undergo preprocessing before being used for fine-tuning. Given the limited size of our datasets, they are partitioned in a 6:1:3 ratio for training, validation, and testing, respectively.

This split aims to improve the reliability of our results as it provides a large proportion of data for testing.

5.1.2.1 Hyperparamter Exploration

The main aim of this experiment is not about finding the optimal hyperparameter values, therefore we will only explore a small set of hyperparameters—learning rate, batch size and Direct Preference Optimization (DPO) β—to gain an intuition for their effects on the fine-tuning process. In doing so, we first establish the constants in our experimental settings. We employ the Adam optimization algorithm (Kingma and Ba, 2014) and a cosine learning rate scheduler (Loshchilov and Hutter, 2016), we also select Low-Rank Adaptation (LoRA) parameters advised by the original paper (Hu et al., 2021). The specifics are detailed in Table 5.1.

Parameter	Value
Learning rate	$\{5\times10^{-3}, \underline{5\times10^{-4}}, 5\times10^{-5}, 5\times10^{-6}\}$
Batch size	$\{2, 3, \underline{4}, 5\}$
Learning algorithm	Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$)
LR scheduler type	Cosine
LR scheduler warmup steps	100
Maximum Training steps	1000
DPO β	$\{0.1, 0.5, \underline{0.9}\}$
LoRA α	16
LoRA r	8
LoRA dropout	0.1

Table 5.1: The hyperparameter values used in the experiment. The learning rate, batch size and DPO β values explored are listed, with their final best values underlined.

In the original paper by Hu et al. (2021), LoRA is only applied to the query and value matrices of the attention mechanism. However, research by He et al. (2021) and Dettmers et al. (2023) found that applying LoRA to more layers can enhance model performance, as this is most similar to full fine-tuning. Thus, we extend LoRA's application to all layers, including the key matrix, the projection layers, and the linear layers.

We first explore the learning rate together with the batch size. The DPO β is set to 0.1, as this is the value used in the original paper (Rafailov et al., 2024). An important point to note is that due to computational and time limitations[1], we only explore these hyperparameters on one user's (user 1) data from the SHPS dataset. Therefore, a crucial assumption this approach makes is that the optimal settings for all data from both datasets are expected to be similar. The values explored and final value that achieved the highest accuracy are listed in Table 5.1. Detailed results from the exploration are specified in Appendix C.1.1.

[1]We explore 3 learning rate values and 3 batch size values, resulting in 9 combinations. If this process was done for each user in each dataset we would need to complete 72 experiments.

Adapters for all users for both datasets are then fine-tuned with our best discovered hyperparameters. In comparing the average accuracy and perplexity of the fine-tuned adapters with a DPO β of 0.1 against the baseline model, it is evident that whilst accuracies increase, perplexity also increases significantly, as seen in Figure 5.1. This suggests the model's probability distribution has shifted too much and may result in degraded performance. We then explore the DPO β to restrict the probability distribution shift.

We retain the best learning rate and batch size found previously and explore DPO β of 0.5 and 0.9 for all datasets. Only these values are explored due to time constraints of our project. Detailed results can be seen in Appendix C.1.2. In Figure 5.1, we observe that the average accuracy for both datasets do not change much for both β of 0.5 and 0.9. However, the average perplexities approaches the baseline most closely at β of 0.9, indicating a smaller distribution shift, and it even obtains a lower value than the baseline model. Therefore, 0.9 is selected as the DPO β and we use adapters fine-tuned with this value for further analysis.

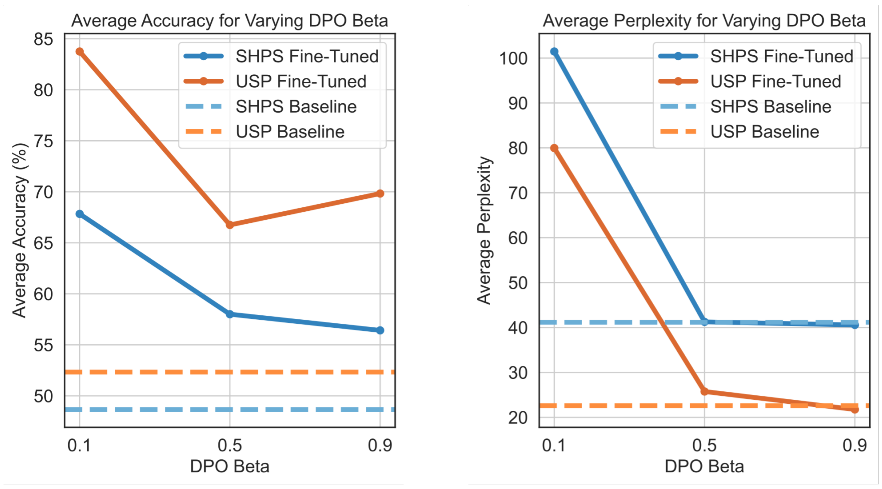

Figure 5.1: The average accuracy (left) and average perplexity (right) of adapters fine-tuned with varying DPO β for both SHPS and USP datasets.

5.1.3 Results and Discussion

Table 5.2 shows the accuracy and perplexity of the baseline model and fine-tuned adapters, for both SHPS and USP datasets. It is immediately evident that the adapters are able to capture user-specific preferences when fine-tuned on corresponding preference data.

From the results, we see that fine-tuned adapters outperform the baseline model in terms of accuracy across all users for both datasets. Specifically, the average accuracy for the SHPS dataset increases from 48.67% for the baseline model to 56.42% for the fine-tuned adapters, improving by 7.75%. On the other hand, for the USP dataset, the average accuracy increases from 52.33% to 69.83%, resulting in an improvement of 17.50%.

Data	User	Fine-Tuned Adapter		Baseline	
		ACC (%)	PPL	ACC (%)	PPL
SHPS	1	**62.00**	<u>37.70</u>	50.00	39.71
	2	**54.00**	<u>38.18</u>	49.67	39.73
	3	**53.33**	<u>44.28</u>	43.33	44.45
	4	**56.33**	42.12	51.67	<u>40.78</u>
	Average	**56.42**	<u>40.58</u>	48.67	41.17
USP	1	**69.33**	<u>22.39</u>	59.00	22.62
	2	**66.33**	24.72	47.33	<u>22.66</u>
	3	**71.00**	<u>19.29</u>	56.67	22.48
	4	**72.67**	<u>20.60</u>	46.33	22.63
	Average	**69.83**	<u>21.75</u>	52.33	22.60

Table 5.2: The accuracies and perplexities of the baseline model and fine-tuned adapters for both SHPS and USP datasets. For each user and on average, the highest accuracies are in **bold** and lowest perplexities are <u>underlined</u>. ACC is accuracy, PPL is perplexity.

The increase in average accuracy for the USP dataset is more than twice as large than the SHPS dataset. This could be because the SHPS dataset reflects a broader range of preferences, due to it being comprised of internet forum questions and responses which are essentially of an open-ended nature, and therefore capturing user preferences in 1000 data samples would be intuitively a hard task. Whereas the USP dataset is engineered to comprise responses reflecting distinct preferences, so the adapter might find it easier to capture specific preferences.

In addition to accuracy, the perplexity also reflects the improved performance of the fine-tuned adapters in comparison to the baseline model. The fine-tuned adapters consistently exhibits lower perplexity across nearly all users for both datasets, with an average reduction from 41.17 to 40.58 in the SHPS dataset and from 22.60 to 21.75 in the USP dataset.

An interesting observation is that the average perplexity for the SHPS dataset is around 41 for both the baseline model and fine-tuned adapters, whereas for the USP dataset, it is almost half this value, at around 22. A difference in perplexity is expected given that the SHPS dataset consists solely of human-written text, which typically exhibits greater variability compared to the language model-generated USP dataset. Moreover, the text generated by the language model likely shares a closer probability distribution with our baseline model than does human-written text, given that it is a product of another large language model.

5.1.3.1 Further Analysis

Recall that each entry in the dataset can be thought of as two parts: prompt combined with chosen response, and prompt combined with rejected response. For simplicity, the two parts will be referred to as just the chosen and rejected responses respectively.

For the data of a particular user in a dataset, we get the average negative log-likelihood (NLL) (see Equation 4.2) for the chosen and rejected response of each entry on both the baseline model and fine-tuned adapters. This is plotted as a density plot. Figure 5.2 shows an example for one user. We analyse the graph for one user as all density plots (see Appendix C.2) display similar features. Furthermore, to gain a more quantitative understanding, for each dataset, we first sum the average NLL of each entry for each user. Then, we find the mean of this sum among the 4 users within the dataset to get what we will refer to as the aggregated average NLL. The results are shown in Figure 5.3.

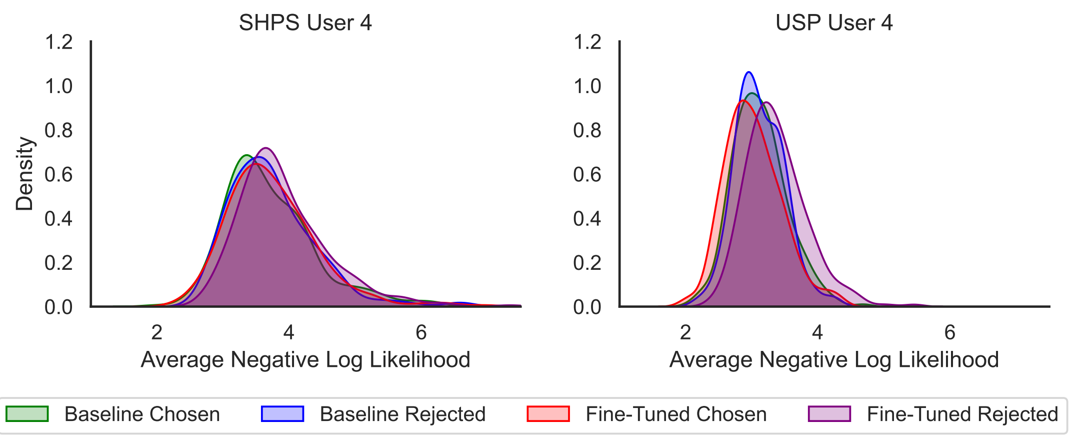

Figure 5.2: The average NLL density plot for user 4 on both SHPS (left) and USP (right), for the baseline model and fine-tuned adapters.

From Figure 5.2, we see that for both SHPS and USP datasets, the baseline model's chosen and rejected distributions have near complete overlap. However, after fine-tuning, the distribution for chosen responses are shifted lower to the left while the distributions for the rejected responses are shifted higher to the right. Quantitatively, Figure 5.3 shows that the difference between the aggregated average NLL for the chosen and rejected responses for both datasets is larger for the fine-tuned adapters in comparison to the baseline models, increasing from -9 to 44 and 9 to 104 for SHPS and USP respectively. This increased gap illustrates the reason for the higher accuracy achieved by the fine-tuned adapters.

The separation of the chosen and rejected distributions after fine-tuning is more evident for the USP dataset than SHPS, as shown in Figure 5.2. The difference in the aggregated average Negative log-likelihood (NLL) between chosen and rejected responses after fine-tuning is more pronounced in the USP dataset, with a value of 104, compared to 44 in the SHPS dataset, as illustrated in Figure 5.3. These observations indicate that the model is more confident in distinguishing between chosen and rejected responses for the USP dataset. This may be because the USP data is less open ended and complex compared to the SHPS data, therefore it allows the model to better capture features for user preferences.

Another point to note is that the distributions for the USP dataset are more tightly clustered between the values of 2 and 4.5, whereas the SHPS dataset tends to spread

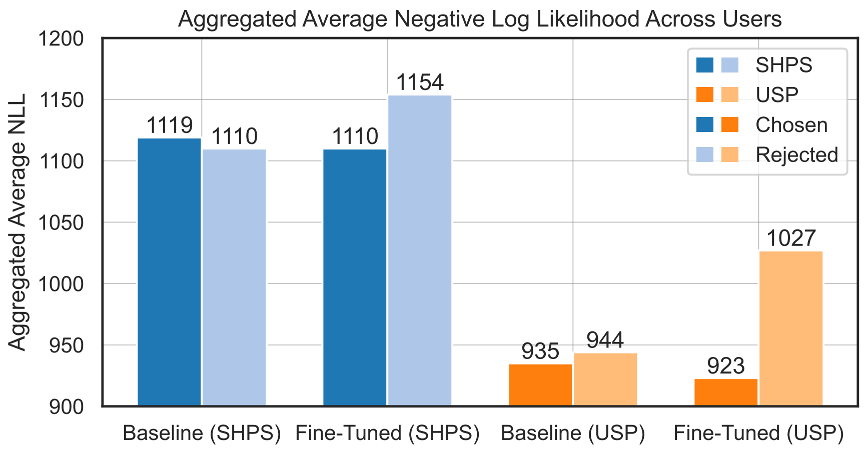

Figure 5.3: The aggregated average NLL for both SHPS (blue) and USP (orange) for chosen (dark) and rejected (light) responses across all users.

more broadly between 2 and 5.5, with a tail extending beyond 7, as depicted in Figure 5.2. Consistently, Figure 5.3 also indicates that the aggregated average NLL for all values for USP is around 10% lower than that for SHPS. This further illustrates the fact that the SHPS data is less predictable than the USP data, due to the same variability and differing probability distribution as previously discussed.

5.2 Merging Adapters for Few-Shot Generalization

5.2.1 Objective

We aim to investigate whether merging our previously fine-tuned adapters can more accurately predict a new unseen user's preferences compared to the baseline model. To determine this, we will compare the performance of the merged adapters against the baseline model on the new user's data. This approach could provide an efficient method for quickly generalizing to a new user's preferences in a few-shot setting, where only a few annotated samples are available. It could also present a better starting point for further fine-tuning to align with individual preferences in comparison to a baseline model. This approach is based on the assumption that users share certain preferences, with some aligning more closely with specific user groups than others.

5.2.2 Experimental Setup

We utilize the fine-tuned adapters for each user for both datasets from experiment 1 in Section 5.1. To select adapters for merging, we evaluate their performance on a small sample of data for a new unseen user. We opt for 30 samples as this intuitively seems as a reasonable minimum to gauge performance of the adapters, even fewer samples may be too noisy for this experimental design.

Given that each of our datasets—the Stanford Human Preferences Subset (SHPS) and User-Specific Preferences (USP)—includes data for four users, there are three adapters fine-tuned on the preferences of the other users for any given user. Any given user can act as the new, unseen user, allowing us to evaluate the performance of each of the three adapters on a small sample of the chosen user's data. Based on their accuracy, we then select adapters to merge. For example, if we select user 1 in the SHPS dataset as the unseen user, we can then select a small sample of that user's training data to assess the accuracies of the adapters fine-tuned on the data from users 2, 3, and 4.

The two adapters that achieve the highest accuracy on unseen data are merged, and similarly, the two with the lowest accuracies are merged. This approach aims to determine the validity of merging based on a few-shot approach, as intuitively the pair of adapters achieving the highest accuracy should outperform the pair with the lowest accuracy. Additionally, we merge all three adapters to model overall population preferences and explore potential improvements in generalization compared to the baseline.

For adapter merging, we assign weights of 1 to each adapter so that they contribute equally to the merged adapter. This weight is set equally for simplicity, it is possible that this is not the optimal setting. Evaluation is conducted on the same 300-sample test sets for all users for both datasets as obtained from the 6:1:3 train, validation and test data split specified in experiment 1 (see Section 5.1.2).

5.2.3 Results and Discussion

Table 5.3 presents the accuracies for 30 samples of unseen data on the fine-tuned adapters. We observe that the adapters generally achieve an accuracy of around 50%, which is comparable to a random guess. However, there are a few examples where this is not the case. For the USP dataset, the adapter fine-tuned on user 1's data achieves a high accuracy of 70.00% on user 3's data. On the other hand, for the SHPS dataset, the adapter fine-tuned on user 2's data shows a low accuracy of 26.67%.

Data	User	Fine-Tuned Adapter Accuracy (%)			
		User 1	User 2	User 3	User 4
SHPS	1	–	**50.00**	**56.67**	30.00
	2	**60.00**	–	46.67	**50.00**
	3	**66.67**	26.67	–	**60.00**
	4	**50.00**	**53.33**	36.67	–
USP	1	–	40.00	**46.67**	**53.33**
	2	**54.33**	–	53.33	**56.67**
	3	**70.00**	50.00	–	**63.33**
	4	**50.00**	**63.33**	43.33	–

Table 5.3: Accuracies of fine-tuned adapters on 30 unseen samples from a new user for both SHPS and USP datasets. The 2 highest and 2 lowest accuracies for each user's data are in **bold** and underlined, respectively. The adapter fine-tuned on the user's own data is scored out.

Data	User	Accuracy (%)							
		Fine-Tuned Adapter				Merged Adapters			Baseline
		User 1	User 2	User 3	User 4	Pair^H	Pair^L	All	
SHPS	1	<u>62.00</u>	51.67	**53.00**	45.00	50.67	**53.00**	47.67	50.00
	2	53.67	<u>54.00</u>	46.67	51.00	**54.33**	53.67	53.00	49.67
	3	52.67	44.00	<u>53.33</u>	51.67	**56.00**	52.00	52.33	43.33
	4	48.33	50.33	49.00	<u>56.33</u>	**54.00**	51.00	50.33	51.67
	Average*	51.56	48.67	50.00	49.22	**53.75**	52.42	50.83	48.67
USP	1	<u>69.33</u>	55.67	54.00	49.67	51.67	51.00	53.67	**59.00**
	2	54.33	<u>66.33</u>	50.00	51.33	**54.67**	51.67	49.67	47.33
	3	51.00	48.33	<u>71.00</u>	52.67	54.00	52.00	53.67	**56.67**
	4	50.67	54.67	47.67	<u>72.67</u>	**56.67**	54.33	48.33	46.33
	Average*	52.00	52.89	50.56	51.22	**54.25**	52.25	51.33	52.33

Table 5.4: Accuracies of the baseline model, fine-tuned adapters and merged adapters for both SHPS and USP datasets. Pair^H denotes the adapter pair with the highest accuracies on 30 samples of unseen data, while Pair^L represents the pair with the lowest accuracies. The accuracy achieved by the adapter fine-tuned on the user's own data is <u>underlined</u>. Average* represents the average accuracy excluding underlined values. The highest accuracy achieved for each user and on average* is in **bold**.

Table 5.4 shows the performance of the adapters merged based on accuracies in Table 5.3. It is clear that merging adapters enhances the average accuracy on unseen data of a new user, compared to the baseline model. For the rest of the analysis we will refer to the merged pair of adapters that achieved the highest accuracies in Table 5.3 as Pair^H, and the merged pair that achieved the lowest accuracies as Pair^L.

Pair^H generally achieves the highest accuracy on unseen data given that we omit accuracies obtained from adapters trained and evaluated on the same user's data. As shown in Table 5.4, this is the case for 5 out of the 8 sets of data. Exceptions include data for user 1 for SHPS, where both the adapter fine-tuned on user 3's data and Pair^L achieve joint highest accuracies. Another exception is in the USP dataset for users 1 and 3, where the baseline model achieves the highest accuracy. Although Pair^H is able to achieve highest accuracy on over half of the sets of data, the significance of this trend should be observed with caution as we only employ 8 sets of data.

The accuracies achieved by Pair^H are generally lower than the accuracies achieved by the adapter that was fine-tuned on data for the specific user. This is not the case for two sets of data: user 2 and user 3 of the SHPS dataset. Surprisingly, Pair^H achieves a 0.33% higher accuracy for user 2 and a 2.77% higher accuracy for user 3. Although the 0.33% increase appears negligible, the 2.77% increase clearly demonstrates significance. Both of these occur within the SHPS dataset, therefore a possible explanation relates to the open ended nature of the responses within the dataset. The data might not be distinct enough for a user to consistently select a preferred response, thus some preferences could be random. Consequently, due to this randomness, another model or adapter might be able to achieve similar or even better accuracy, potentially by chance.

For the SHPS dataset, the accuracies of Pair^H are generally comparable to those of the adapter fine-tuned on data for the specific user, underlined in Table 5.4, ranging between 50% and 62%. However, in the USP dataset, Pair^H achieves significantly lower accuracies, generally in the 50s, compared to the adapters fine-tuned on data for the specific user, underlined in Table 5.4, which achieve around 70%. This suggests that merging adapters is less beneficial for the USP dataset compared to SHPS. Given that the USP dataset contains responses that reflect more distinct preferences, the fine-tuned adapters may become overly specific to the data of users they were trained on, leading to more noise when merged. To achieve accuracies in the 70% range, it might be necessary to merge adapters that are even more aligned with each other in terms of user preferences.

Another observation we make is that the average accuracy of Pair^H, with 53.75% for SHPS and 54.25% for USP, is higher than the average accuracy of fine-tuned adapters which are not tuned on the user of interest's data, 51.56%, 48.67%, 50.00% and 49.22% for SHPS and 52.00%, 52.89%, 50.56% and 51.22% for USP. This proves that merging adapters which are somewhat aligned to a user's preferences results in better performance than an individual adapter which shows signs of alignment, perhaps because two aligned adapters are able to complement each other to better model the user of interest's preferences.

Lastly, we observed that Pair^H achieves a higher average accuracy compared to Pair^L for both datasets, with 53.75% to 52.42% for SHPS and 54.25% to 52.25% for USP. This shows that merging two more aligned adapters achieves higher accuracy than two less aligned adapters and thus validates our few-shot approach. Furthermore, both Pair^H and Pair^L achieve a higher average accuracy than merging all adapters, which is 50.83% for SHPS and 51.33% for USP. However, two cases exist where merging all adapters performs worse than Pair^H but better than Pair^L. For SHPS user 3, Pair^H, Pair^L and merging all adapters achieves 56.00%, 52.00% and 52.33% respectively. For USP user 3, Pair^H, Pair^L and merging all adapters achieves 54.00%, 52.00% and 53.67% respectively. This shows signs that merging all adapters to generalize to overall population preferences might be possible. Here in our experiments, merging all adapters results in a lower average accuracy possibly because we only consider 3 adapters, which could introduce additional noise instead.

5.3 How Much Data Do We Need?

5.3.1 Objective

We aim to investigate how the quantity of data utilized in the fine-tuning process affects the adapter's ability to capture preferences for individual users. Understanding this relationship could be important in a case involving many users, where our goals would be to maximize the accuracy for each user while reducing both the training budget and the need to collect and store extensive data for each individual.

5.3.2 Experimental Setup

The fine-tuned adapters from experiment 1 (see Section 5.1) were trained using 600 samples, with our datasets—Stanford Human Preferences Subset (SHPS) and User-Specific Preferences (USP)—divided in a 6:1:3 ratio for train, validation, and test sets. We maintain this ratio to ensure consistency in the validation and testing segments. The initial 400 and 200 of the 600 samples in the training set are selected and used for fine-tuning the baseline model. otherwise, we adhere to the same experimental setup as in experiment 1 (see Section 5.1.2), including the hyperparameter settings listed in Table 5.1.

5.3.3 Results and Discussion

Table 5.5 shows the accuracy and perplexity of adapters fine-tuned on different numbers of training samples. It is evident that a larger number of samples leads to higher accuracy for the SHPS dataset, but not necessarily for the USP dataset.

Data	User	Fine-Tuned Adapter						Baseline	
		600		400		200			
		ACC (%)	PPL	ACC (%)	PPL	ACC (%)	PPL	ACC (%)	PPL
SHPS	1	**62.00**	37.70	59.67	42.34	53.33	44.58	50.00	39.71
	2	54.00	38.18	**55.00**	41.13	49.67	40.50	49.67	39.73
	3	**53.33**	44.28	46.00	44.00	46.00	45.36	43.33	44.45
	4	56.33	42.14	**58.33**	39.73	54.67	40.98	51.67	40.78
	Average	**56.42**	40.58	54.75	41.80	50.92	42.86	48.67	41.17
USP	1	69.33	22.39	**76.67**	19.26	58.00	23.24	59.00	22.62
	2	66.33	24.72	**68.33**	21.11	56.00	25.38	47.33	22.66
	3	**71.00**	19.29	70.00	22.28	60.00	24.98	56.67	22.48
	4	**72.67**	20.60	67.67	21.62	56.00	25.95	46.33	22.63
	Average	69.83	21.75	**70.67**	21.07	57.50	24.89	52.33	22.60

Table 5.5: Accuracies of the baseline model and adapters fine-tuned on 600, 400 and 200 samples of data for both SHPS and USP datasets. For each user's data and on average, the highest accuracies are in **bold** and lowest perplexities are underlined. ACC is accuracy, PPL is perplexity.

The best accuracies and perplexities across each dataset vary between the adapter fine-tuned on 600 samples and the one on 400 samples. For the SHPS dataset, the 600-sample adapter achieves an average accuracy of 56.42%, while the 400-sample adapter has 54.75%. In contrast, for the USP dataset, the 400-sample adapter achieves a higher average accuracy of 70.67% compared to 69.83% for the 600-sample adapter. This suggests that 400 samples is sufficient for the adapter to capture the features of the USP data, likely due to the distinct preferences within this dataset. On the other hand, the SHP dataset requires more samples as its accuracy continues to improve with increased sample size, possibly due to the open-ended and more complex nature of the dataset.

The average accuracy of the 200-sample adapters is lower than both the 600 and 400-sample adapters for both SHPS and USP, although it still surpasses the baseline model. For both datasets, the accuracy of the 200-sample adapters are closer to that of the baseline model than to the 400 and 600-sample adapters, suggesting that 200 samples is insufficient to capture features in the data. This is supported by the average perplexity of the 200-sample adapter. The average perplexity is relatively consistent across all models, except for the 200-sample adapters, which is meaningfully higher than those of the baseline, 600-sample and 400-sample adapters. This suggests that using only 200 samples for fine-tuning shifts the probability distribution of the model too much, potentially degrading its language capabilities.

To conclude, it appears that for both the SHPS and USP datasets, using around 200 training samples is generally insufficient, as evidenced by the lower accuracy and higher perplexity. For less complex datasets like USP, approximately 400 training samples achieves meaningful improvement, however, for more complex datasets like SHPS, which shows a consistent improvement in accuracy with an increase in number of training samples, further research is needed to determine how this trend progresses.

5.3.3.1 Further Analysis

We again utilize the average negative log-likelihood (NLL) and aggregated average NLL, as defined in Section 5.1.3.1, to analyze the chosen and rejected responses for adapters fine-tuned on 600, 400, and 200 samples for both SHPS and USP datasets. Specifically, we again analyse the density plot for one user as all plots (see Appendix D.1) display similar features.

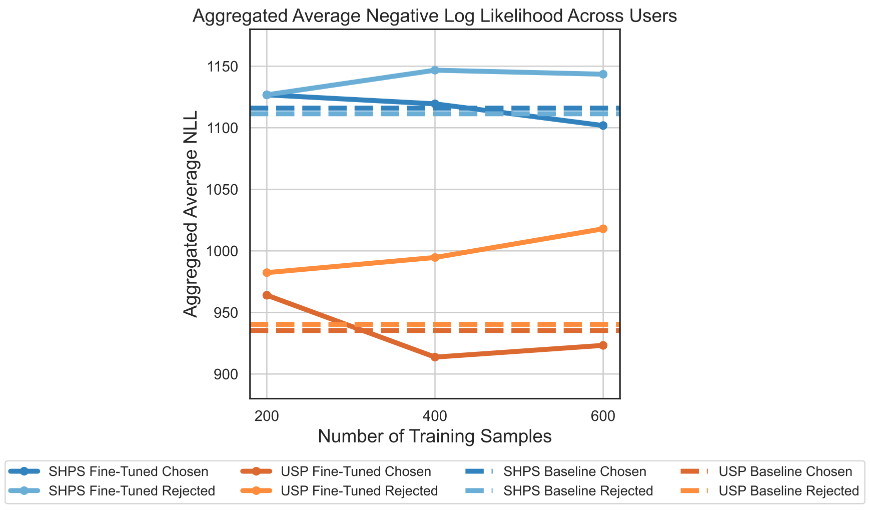

Figure 5.4: The aggregated average NLL across all users, with varying training sample sizes, for both SHPS (blue) and USP (orange) for chosen (dark) and rejected (light) responses.

Figure 5.4 shows that as the number of training data increases, the difference in aggregated average NLL between chosen and rejected responses for fine-tuned adapters on both SHPS and USP datasets also increases. This suggests that the models becomes more certain in their predictions. Notably, the gap for USP data between chosen and rejected responses in fine-tuned adapters is much larger than that for SHPS data. This suggests that features in USP data may be more easily capture by the adapters compared to SHPS, which aligns with the nature of the datasets-USP data reflects distinct preferences, whereas SHPS is more open-ended and complex.

Figure 5.5 further supports this finding. It shows that the distributions among varying number of samples for SHPS's chosen and rejected responses do not tend to change much. On the other hand, for USP data, there is a noticeable shift: the distribution of chosen responses tends to move towards 0 as the number of samples increases, while the distribution of rejected responses shifts away from 0 with more samples.

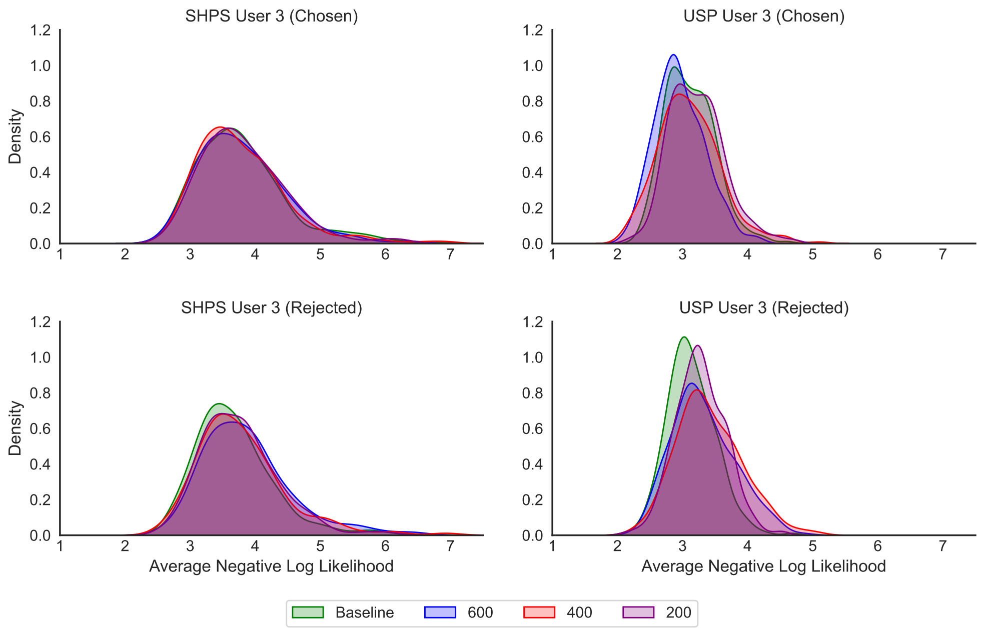

Figure 5.5: The average NLL density plots for user 3, with varying number of training samples of both SHPS (left) and USP (right), for chosen (top) and rejected responses (bottom).

The aggregated average NLL for chosen responses in the SHPS dataset for fine-tuned adapters strictly decreases with an increasing number of training samples, as seen in 5.4. While the aggregated average NLL for rejected responses also seems to decrease going from 400 to 600 training samples, the decrease is not as steep. If this trend continues, it suggest that the model would be more confident in differentiating between chosen and rejected responses, potentially leading to a higher accuracy. This observation is consistent with the accuracy trends reported in Table 5.5 and previous discussion.

On the other hand, for the USP adapters, the aggregated average NLL for chosen responses increases as the number of training samples increases from 400 to 600. The aggregated average NLL for rejected responses also increases at a similar rate. If this trend continues with an increase in training data, the gap between the aggregated average NLL for chosen and rejected responses would remain consistent. Therefore the accuracy is likely to remain at a similar value. This finding aligns with previous discussions based on accuracies in Table 5.5.

5.4 Summary

This chapter examines the alignment of a large language model (LLM), Llama 2-Chat 7B Touvron et al. (2023a), to user-specific preferences using the Stanford Human Preferences Subset (SHPS) and User-Specific Preferences (USP) datasets. We conducted several experiments and obtained the following findings:

- We conducted experiments to assess the effectiveness of an LLM in capturing user-specific preferences across both datasets. Our results show that while the **LLM effectively capture preferences**, its **performance is lower on the SHPS dataset compared to the USP** dataset. Additionally, we observed that the data distribution in USP aligns more closely with the base model than SHPS, and after fine-tuning, the gap between chosen and rejected distributions in USP grows more significantly than in SHPS. These are all attributed to the more complex nature of SHPS.
- In our experiments focusing on few-shot generalization capabilities, we found that **merging two highly aligned adapters yields better results than merging two less aligned adapters. Additionally, both generally outperform merging all adapters.** Furthermore, compared to respective baselines, **merging yields smaller improvements for the USP dataset than for SHPS.** This suggests that the characteristics of the dataset significantly affect performance of merged adapters. Simpler datasets such as USP may exhibit worse performance, potentially due to overfitting to specific user preferences.
- We also investigated the amount of data required to meaningfully capture user-specific preferences. Our results show that using 200 samples provides moderate improvements, but increasing to **400 samples results in the most significant improvement**. Subsequent increases to 600 samples offer diminishing returns. Additionally, while **more data generally enhances accuracy for the SHPS dataset, this is not necessarily the case for the USP dataset**. Furthermore, the distributions for USP data are affected more by the training sample size than the SHPS data. These are all again possibly due to SHPS's inherent complexity.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this project we presented an exploration into aligning large language models (LLMs), specifically Llama 2 (Touvron et al., 2023a) with user-specific preferences. Low-Rank Adaptation (LoRA) (Hu et al., 2021) is utilized to capture user-specific preferences. To facilitate exploration, we introduced 2 new datasets: the Stanford Human Preferences Subset (SHPS) which contains preferences of 4 users on 1000 entries of human-written text, and User-Specific Preferences (USP) which contains preferences of 4 users on 1000 entries of language model generated text.

Our findings show that through fine-tuning with LoRA, we can effectively capture preferences of users in both datasets. Though, a higher improvement in average accuracy in comparison to the baseline is observed for USP than SHPS. We reason that this is due to the nature of the datasets. SHPS is more open-ended and complex than USP, and therefore harder for the user to discern a preferred responses. This is supported by our initial analysis on agreement among user preferences which showed that there exists distinct correlations among users in the USP dataset compared to more random relationships as seen in SHPS.

Additionally, we show that merging adapters from different users can improve accuracy above the baseline for a new unseen user. Notably, this is more effective for SHPS data than USP. As discussed, USP data contains more distinct preferences than SHPS, and thus adapters may become more specialized, resulting in loss in accuracy when generalizing to a new user. Merging all adapters also generally perform worse than merging the two most aligned or two least aligned adapters, due to an increase in possible conflicting preferences.

Lastly, our analysis of training sample sizes show that while 200 samples are sufficient for minor improvements in accuracy, at least 400 samples are necessary for meaningful improvements. The USP dataset reaches the highest average accuracy at 400 samples, whereas the SHPS dataset continues to show a consistent increase in accuracy with additional samples. This again indicates that SHPS is more complex in comparison to USP due to its open-ended nature, and requires more data for increased accuracy.

6.2 Limitations and Future Work

Our findings are based on our two datasets: the Stanford Human Preferences Subset (SHPS) and User-Specific Preferences (USP). However, there are several issues with these datasets that may impact the robustness of our conclusions:

- **Quantity:** The size of both datasets is relatively small, with only 4 users, which necessitates caution in interpreting our findings. In particular, our adapter merging experiments were limited to only three adapters, making the results potentially non-conclusive. Future work could consider collecting more data or utilizing other datasets with a larger number of entries for user-specific preferences.
- **Similarity:** As discussed in Section 3.2.2, some instructions within the USP dataset are very similar, raising concerns about the diversity of the data. Although research has shown that creativity is inherently limited in LLMs (Bunescu and Uduehi, 2019; Shanahan, 2023), Zhao et al. (2024) show that there are still many factors that could be investigated to enhance creativity. These include the type of model architecture, the prompts received by the model, and the system prompts used. Future work could explore these factors to obtain more diverse data.
- **Complexity:** Our interpretation of results relies on the assumption that the SHPS data is more open-ended and complex, and thus presents challenges in discerning user preferences—this assumption has yet to be proven. Future work could investigate this further, exploring the claims made in this work. Additionally, it would be beneficial to explore the specific relationship between the complexity of preference data and the effectiveness of LLMs in learning user-specific preferences, also considering how the number of samples affects this relationship.

Compiling a large dataset with annotations is not only costly but also time-consuming. The process of generating data and collecting responses for our small dataset took over 60 hours. Future work could explore further utilizing the Self-Instruct framework to generate annotated data. Serapio-García et al. (2023) show that LLMs are able to reflect human personalities, thus, we can explore adding seed tasks and prompts to guide the LLM in generating annotated data based on specific personality traits. Adapters can then be fine-tuned using this annotated data to serve as basic starting points for merging to allow for a better approximation of a new user's preferences. This can then also act as the starting point to be further fine-tuned to the preferences of this new user.

Lastly, in this work we explored adapter merging using equal weights for simplicity. Future work could explore using different weights or learnable weights, as adapters may contribute differently to the model's performance on unseen data. Furthermore, we selected adapters to merge based on their accuracy with 30 unseen samples. We observed that the accuracies achieved by the adapters on these samples were around 50%. Therefore, the robustness of this approach may seem weak. Future work could further explore the few-shot setting and formulate a robust framework.

Bibliography

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*, 2020.

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, et al. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'24). Association for Computing Machinery, New York, NY, USA*, pages 317–335, 2024.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. Sparks of artificial general intelligence: Early experiments with gpt-4, 2023.

Razvan C Bunescu and Oseremen O Uduehi. Learning to surprise: A composer-audience architecture. In *ICCC*, pages 41–48, 2019.

Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Freedman, Tomasz Korbak, David Lindner, Pedro Freire, Tony Wang, Samuel Marks, Charbel-Raphaël Segerie, Micah Carroll, Andi Peng, Phillip Christoffersen, Mehul Damani, Stewart Slocum, Usman Anwar, Anand Siththaranjan, Max Nadeau, Eric J. Michaud, Jacob Pfau, Dmitrii Krasheninnikov, Xin Chen, Lauro Langosco, Peter Hase, Erdem Bıyık, Anca Dragan, David Krueger, Dorsa Sadigh, and Dylan Hadfield-Menell. Open problems and fundamental limitations of reinforcement learning from human feedback, 2023.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In Steven Bethard, Marine Carpuat, Marianna Apidianaki, Saif M. Mohammad, Daniel Cer, and David Jurgens, editors, *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/S17-2001. URL `https://aclanthology.org/S17-2001`.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: An open platform for evaluating llms by human preference, 2024.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

ECDF. Edinburgh compute and data facility. `www.ecdf.ed.ac.uk`. Accessed: 20 April 2024.

Kawin Ethayarajh, Yejin Choi, and Swabha Swayamdipta. Understanding dataset difficulty with $\mathcal{V}$-usable information. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 5988–6008. PMLR, 17–23 Jul 2022. URL `https://proceedings.mlr.press/v162/ethayarajh22a.html`.

Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A Smith. Realtoxicityprompts: Evaluating neural toxic degeneration in language models. *arXiv preprint arXiv:2009.11462*, 2020.

Richard W Hamming. *Coding and information theory*. Prentice-Hall, Inc., 1986.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*, 2021.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR, 2019.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):S63–S63, 1977.

Jiaming Ji, Mickel Liu, Juntao Dai, Xuehai Pan, Chi Zhang, Ce Bian, Chi Zhang, Ruiyang Sun, Yizhou Wang, and Yaodong Yang. Beavertails: Towards improved safety alignment of llm via a human-preference dataset, 2023.

Junsol Kim and Byungkyu Lee. Ai-augmented surveys: Leveraging large language models and surveys for opinion prediction, 2024.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Nikita Kitaev and Dan Klein. Constituency parsing with a self-attentive encoder. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1249. URL `https://aclanthology.org/P18-1249`.

Nikita Kitaev, Steven Cao, and Dan Klein. Multilingual constituency parsing with self-attention and pre-training. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3499–3505, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1340. URL `https://aclanthology.org/P19-1340`.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and

Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. URL `https://aclanthology.org/2021.emnlp-demo.21`.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

Chin-Yew Lin and Franz Josef Och. Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 605–612, Barcelona, Spain, July 2004. doi: 10.3115/1218955.1219032. URL `https://aclanthology.org/P04-1077`.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. Peft: State-of-the-art parameter-efficient fine-tuning methods. `https://github.com/huggingface/peft`, 2022.

L. J. Miranda. Study notes on parameter-efficient finetuning techniques. Blog Post, 2023. URL `https://ljvmiranda921.github.io/notebook/2023/05/01/peft/#pfeiffer2023modular`. Available at `https://ljvmiranda921.github.io/notebook/2023/05/01/peft/#pfeiffer2023modular`.

Saif M. Mohammad and Graeme Hirst. Distributional measures of semantic distance: A survey, 2012.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback, 2022.

OpenAI. Gpt-4 technical report, 2024.

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. Training language models to follow instructions with human feedback, 2022.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

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

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in*

Natural Language Processing. Association for Computational Linguistics, 11 2019. URL `https://arxiv.org/abs/1908.10084`.

Neal Richardson, Ian Cook, Nic Crane, Dewey Dunnington, Romain François, Jonathan Keane, Dragoș Moldovan-Grünfeld, Jeroen Ooms, Jacob Wujciak-Jens, and Apache Arrow. *arrow: Integration to 'Apache' 'Arrow'*, 2024. URL `https://github.com/apache/arrow/`. R package version 16.0.0, https://arrow.apache.org/docs/r/.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. de rumelhart and j. mcclelland. vol. 1. 1986. *Biometrika*, 71:599–607, 1986.

Alireza Salemi, Sheshera Mysore, Michael Bendersky, and Hamed Zamani. Lamp: When large language models meet personalization, 2024.

Shibani Santurkar, Esin Durmus, Faisal Ladhak, Cinoo Lee, Percy Liang, and Tatsunori Hashimoto. Whose opinions do language models reflect? In *International Conference on Machine Learning*, pages 29971–30004. PMLR, 2023.

Abulhair Saparov, Richard Yuanzhe Pang, Vishakh Padmakumar, Nitish Joshi, Seyed Mehran Kazemi, Najoung Kim, and He He. Testing the general deductive reasoning capacity of large language models using ood examples, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Greg Serapio-García, Mustafa Safdari, Clément Crepy, Luning Sun, Stephen Fitz, Peter Romero, Marwa Abdulhai, Aleksandra Faust, and Maja Matarić. Personality traits in large language models, 2023.

Deven Shah, H Andrew Schwartz, and Dirk Hovy. Predictive biases in natural language processing models: A conceptual framework and overview. *arXiv preprint arXiv:1912.11078*, 2019.

Murray Shanahan. Talking about large language models, 2023.

Taylor Sorensen, Jared Moore, Jillian Fisher, Mitchell Gordon, Niloofar Mireshghallah, Christopher Michael Rytting, Andre Ye, Liwei Jiang, Ximing Lu, Nouha Dziri, Tim Althoff, and Yejin Choi. A roadmap to pluralistic alignment, 2024.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Longyue Wang, Chenyang Lyu, Tianbo Ji, Zhirui Zhang, Dian Yu, Shuming Shi, and Zhaopeng Tu. Document-level machine translation with large language models, 2023a.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions, 2023b.

Leandro Von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, and Shengyi Huang. Trl: Transformer reinforcement learning. `https://github.com/huggingface/trl`, 2020.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. Resolving interference when merging models. *arXiv preprint arXiv:2306.01708*, 2023.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning, 2023.

Yunpu Zhao, Rui Zhang, Wenyi Li, Di Huang, Jiaming Guo, Shaohui Peng, Yifan Hao, Yuanbo Wen, Xing Hu, Zidong Du, Qi Guo, Ling Li, and Yunji Chen. Assessing and understanding creativity in large language models, 2024.

Appendix A

Data Generation

A.1 Seed Tasks

Table A.1: The 30 seed tasks used in our data generation procedure. Each instruction is paired with an output that contains two responses A and B.

ID	Instruction	Output
1	I love hiking, but the weather is unpredictable. Can you suggest a way to experience nature without leaving my home?	A) Bring nature indoors by creating a mini indoor garden with plants that thrive in low-light conditions. You can even add special lighting and a small water feature for a more realistic experience. B) Use virtual reality technology to go on a hike from the comfort of your own living room. Feel like you're really in nature with interactive sights and sounds.
2	I'm on a budget but still want to try being creative. How can I make decorations at home without spending a lot of money?	A) Upcycle old items by painting them or giving them a new purpose. For example, turn mason jars into candle holders or create a wall hanging using leftover fabric and a embroidery hoop. B) Shop at thrift stores or garage sales for cheap yet unique decorations. You never know what hidden gems you might find!
3	I want to surprise my partner with a romantic date, how can I plan something special for the evening without relying on cliché ideas?	A) Cook a fancy dinner together at home using a new recipe, or try out a unique cuisine you've never had before. Eating together can be a fun way to bond and try something new. B) Go to a local escape room or solve a mystery box together. These activities provide a unique and exciting experience that will surely surprise your partner.

Continued on next page

Table A.1 Continued from previous page

ID	Instruction	Output
4	I'm interested in learning a new language, but don't have much time during the day. How can I incorporate language practice into my daily routine?	A) Listen to language learning podcasts or audiobooks while doing tasks like cooking or commuting. This will help you absorb the language without taking up extra time. B) Use language learning apps that send daily vocabulary or grammar exercises to your phone. You can easily complete these exercises during short breaks throughout the day.
5	I have a big presentation tomorrow and need some help calming my nerves. What are some relaxation techniques I can do in the comfort of my own home?	A) Practice deep breathing exercises or gentle yoga poses to relax your body and mind. You can find videos or guided meditations online to help you through the process. B) Use aromatherapy by diffusing calming scents like lavender or chamomile. The scent alone can have a positive effect on your mood and reduce stress.
6	I want to learn how to cook - suggest a recipe for me.	A) You could start by trying a simple and delicious recipe like homemade spaghetti aglio e olio. This classic Italian dish features spaghetti tossed with garlic, olive oil, red pepper flakes, and parsley for a flavorful and satisfying meal that is easy to make. B) Experiment with making a basic stir-fry with your favorite vegetables, protein, and sauce. Stir-fries are quick, versatile, and a great way to practice your cooking skills while customizing the dish to suit your tastes.
7	I love reading but have trouble finding time to sit down with a book. What are some creative ways to fit reading into my daily routine?	A) Read while doing other activities, like listening to an audiobook while cooking or reading an e-book during your commute. This allows you to multitask and still enjoy a good book. B) Set a reading goal for yourself, such as reading a certain number of pages a day or finishing a book by a certain date. This can help motivate you and make reading a regular habit.

Continued on next page

Table A.1 Continued from previous page

ID	Instruction	Output
8	I'm feeling overwhelmed with work and need a way to destress. How can I incorporate self-care into my busy schedule?	A) Take a break and go for a walk or do some gentle stretches. Moving your body can help reduce stress and improve your mood. B) Use a journal or meditation app to reflect on your thoughts and emotions. Taking a few minutes to slow down and check in with yourself can make a big difference in your overall well-being.
9	I'm feeling bored at home, give me some ideas.	A) Create a relaxing ambiance by lighting scented candles, playing calming music, and treating yourself to a DIY spa day with a bubble bath, face masks, and indulgent skincare routine. Unwind with a good book, journaling, or practicing mindfulness to soothe your mind and body. B) Engage in virtual social activities like video calls with friends or virtual game nights to stay connected and entertained. Explore online museums, virtual tours, or live-streamed concerts and events for a dose of culture and entertainment from the comfort of your home.
10	What's a peaceful morning routine to start the day with calm?	A) Begin your peaceful morning with a gentle stretching or yoga session to awaken your body and mind. Practice deep breathing exercises or mindfulness meditation to promote inner calm and clarity before starting your day. B) Savor a warm cup of herbal tea or coffee while enjoying a quiet moment of reflection or journaling. Create a serene ambiance with soft lighting, soothing music, or nature sounds to set a tranquil atmosphere and ease into the day peacefully.
11	I want to get active, what sports could I play?	A) Consider trying sports like tennis, swimming, or hiking to stay active and enjoy the outdoors. These activities can help improve cardiovascular fitness, strength, and flexibility while providing a fun and social way to exercise. B) Explore team sports such as basketball, soccer, or volleyball for a more competitive and dynamic workout. Joining a local recreational league or group can help you stay motivated and engaged while improving your athletic skills and overall fitness level.

Continued on next page

Table A.1 Continued from previous page

ID	Instruction	Output
12	Suggest the best way to learn a new language.	A) Immerse yourself in the language by watching movies, listening to music, and reading books in the target language to improve vocabulary and comprehension. Practice speaking with native speakers through language exchange programs or online conversation groups to enhance your conversational skills and pronunciation. B) Use language learning apps like Duolingo, Babbel, or Rosetta Stone to build a strong foundation in grammar, vocabulary, and sentence structure.
13	I'm going on a road trip soon and want to make the most of my journey. What are some activities I can do while on the road to make the experience more enjoyable?	A) Play road trip games like "I Spy," the license plate game, or 20 questions. These can keep you and your travel companions engaged and entertained during a long drive. B) Listen to podcasts or audiobooks to learn something new or get lost in a story while on the road. This can also be a great way to pass the time and keep you relaxed.
14	I have trouble falling asleep and want to try a more natural approach to improve my sleep quality. Any suggestions?	A) Use aromatherapy by diffusing calming scents like lavender or chamomile in your bedroom. These scents can promote relaxation and improve sleep. B) Try herbal remedies like valerian root or melatonin supplements. These natural sleep aids can be just as effective as prescribed medications without the potential side effects.
15	Suggest a genre of music to listen to.	A) Why not try listening to some relaxing classical music to help you unwind and focus? Classical music can be soothing and beneficial for concentration, relaxation, and stress relief. B) You might enjoy exploring the upbeat and energetic sounds of jazz music. Jazz can uplift your mood, inspire creativity, and provide a vibrant and dynamic soundtrack to your day.

Continued on next page

Table A.1 Continued from previous page

ID	Instruction	Output
16	Recommend me a hot beverage to drink.	A) How about enjoying a cozy cup of chamomile tea? Chamomile tea is known for its calming and soothing properties, making it perfect for relaxation and unwinding. B) You could savor a rich and indulgent hot chocolate topped with whipped cream or marshmallows for a comforting and decadent treat.
17	What sports can I watch on TV?	A) Consider watching exciting basketball games for fast-paced action and thrilling moments. Basketball is a popular sport known for its competitiveness and entertainment value. B) An option is to tune in to competitive soccer matches for intense gameplay and strategic plays. Soccer, also known as football in many countries, offers a dynamic and engaging viewing experience for sports enthusiasts.
18	What should I order at the bar?	A) You might enjoy ordering a classic cocktail like a margarita or mojito for a refreshing and flavorful drink. These popular cocktails are perfect for sipping and socializing at the bar. B) Consider trying a local craft beer or a trendy hard seltzer for a modern and distinctive beverage choice. Craft beers and hard seltzers offer a variety of flavors and options to suit your taste preferences.
19	Recommend a drink to pair with my meal.	A) Consider pairing your meal with a crisp and refreshing white wine like a Sauvignon Blanc or Chardonnay. White wines can complement a variety of dishes, enhancing flavors and providing a balanced dining experience. B) You might enjoy a smooth and medium-bodied red wine such as a Pinot Noir or Merlot to accompany your meal. Red wines can offer depth and richness to dishes, making them a versatile choice for pairing with different types of cuisine.

Continued on next page

Table A.1 Continued from previous page

ID	Instruction	Output
20	What hobbies can I adopt that are both fun and eco-friendly?	A) Why not try hiking or nature photography as eco-friendly hobbies that allow you to appreciate the outdoors and capture the beauty of the environment? Take a leisurely hike through local trails, parks, or nature reserves to connect with nature and enjoy fresh air and exercise. B) Explore urban gardening or indoor plant care as a hobby that promotes sustainability and green living. Cultivate your own indoor garden with houseplants, herbs, or succulents, and practice eco-friendly gardening techniques like composting, water conservation, and using natural fertilizers.
21	I'm fascinated by space. How can I spend an evening exploring the cosmos from home?	A) Set up a stargazing spot in your backyard or by a window. Use a telescope or binoculars to observe the moon, stars, and visible planets. B) Dive into the universe virtually through online planetarium websites or apps. Explore galaxies, nebulas, and constellations in stunning detail right from your couch.
22	I want to spend a day in silence. What activities can I do?	A) Engage in silent meditation or yoga to connect with your inner self. Spend time in nature, observing the beauty around you without the need for words. B) Pursue silent hobbies such as painting, reading, or crafting. These activities allow for self-expression and relaxation without verbal communication.
23	How can I take my taste buds on a world tour from my kitchen?	A) Dedicate each week to exploring a different cuisine, cooking traditional dishes and pairing them with native music. B) Host a virtual international potluck with friends, where each person prepares and shares a recipe from a different country.
24	I want to diversify my reading list. Any suggestions?	A) Explore works by authors from various cultures and backgrounds to gain new perspectives. Include genres you typically don't read to broaden your literary experience. B) Join a book club that focuses on diversity in literature or use platforms that recommend books based on different themes and author backgrounds.

Continued on next page

Table A.1 Continued from previous page

ID	Instruction	Output
25	What are some sustainable commuting options?	A) Consider biking, walking, or using public transport to reduce carbon emissions; explore carpooling with coworkers or neighbors. B) Investigate the feasibility of electric or hybrid vehicles; telecommute if your job permits to minimize travel.
26	How can I have an adventure in my own city?	A) Plan a 'tourist day' to visit local landmarks, museums, or parks you've never been to; try a new restaurant or cuisine each month. B) Join a local hobby or sports group; participate in city-wide scavenger hunts or themed walks to discover hidden gems.
27	How can I get involved in community service?	A) Research local charities or community centers and sign up to volunteer; organize a neighborhood clean-up or food drive. B) Offer your skills to non-profits, such as tutoring, web design, or legal advice; join community boards or local action groups to contribute to local decision-making.
28	I have a free weekend coming up. What's a fulfilling project I can tackle?	A) Organize a space in your home that's been neglected; declutter, donate, and redesign for a fresh start. B) Dive into a creative project like painting a piece of art, crafting a scrapbook, or writing a short story.
29	How can I make my wardrobe more sustainable?	A) Start by decluttering your closet and donating or upcycling clothes you no longer wear. Invest in timeless, high-quality pieces made from sustainable materials that will last longer and have a lower environmental impact. B) Explore second-hand and vintage clothing shops for unique finds that reduce waste and promote a circular fashion economy. Consider renting clothes for special occasions or swapping clothes with friends to refresh your wardrobe sustainably.

Continued on next page

Table A.1 Continued from previous page

ID	Instruction	Output
30	How can I contribute to volunteer efforts from home?	A) Consider virtual volunteering opportunities such as online tutoring, writing letters to isolated individuals, or participating in virtual fundraising events to support causes you care about. Use your skills and time to make a positive impact from the comfort of your home. B) Raise awareness and advocate for social issues through social media campaigns, sharing educational resources, or organizing virtual community events. By using your voice and platforms to spread awareness, you can contribute to important causes and make a difference in your community and beyond.
31	What should I do on a rainy day?	A) A great idea for a rainy day is to bake some delicious treats like cookies, brownies, or a warm loaf of bread. The comforting aroma of baking will fill your home and lift your spirits on a gloomy day. B) Embrace the rain and go for a walk in the rain with a colorful umbrella. Enjoy the calming sound of raindrops and the fresh scent of wet earth as you take a leisurely stroll in nature.

A.2 Prompt Template

The prompt template we use to generate new instructions and responses is adapted from the Stanford Alpaca project (Taori et al., 2023). Sections removed are marked in red, retained sections are in black, and added sections are highlighted in green, as shown below.

Prompt Template

You are asked to come up with a set of 20 diverse task instructions. These task instructions will be given to a GPT model and we will evaluate the GPT model for completing the instructions use the GPT model's completions to create a Reinforcement Learning from Human Feedback dataset.

Here are the requirements:

1. The output should contain two responses A and B, to reflect personal preferences of two different users with opposite preferences.
2. It is curcial to think outside the box to come up with random and unexpected instructions.
3. The instructions must ask a vague, open-ended question.
4. The instructions must not ask for the AI assistant's own preferences
5. The instructions must not ask general knowledge questions
6. Try not to repeat the verb for each instruction to maximize diversity.
7. The language used for the instruction also should be diverse. For example, you should combine questions with imperative instrucitons.
8. A GPT language model should be able to complete the instruction. For example, do not ask the assistant to create any visual or audio output. For another example, do not ask the assistant to wake you up at 5pm or set a reminder because it cannot perform any action.
9. The instructions should be in English.
10. The instructions should be 1 to 2 sentences long. Either an imperative sentence or a question is permitted.
11. The output should be an appropriate response to the instruction and the input. Make sure the output is less than 100 words.
12. The type of instructions should be diverse. The list should include diverse types of tasks like open-ended generation, classification, editing, etc.
13. You should generate an appropriate input to the instruction. The input field should contain a specific example provided for the instruction. It should involve realistic data and should not contain simple placeholders. The input should provide substantial content to make the instruction challenging but should ideally not exceed 100 words.
14. Not all instructions require input. For example, when a instruction asks about some general information, "what is the highest peak in the world", it is not necessary to provide a specific context. In this case, we simply put " <noinput>" in the input field.

List of 20 tasks:

Appendix B

Methodology

B.1 Code Frameworks and Versions

The specific code frameworks used in this project and their versions are presented in Table B.1.

Framework	**Version**
PyTorch	2.2.1+cu121
Transformers	4.39.1
PEFT	0.10.0
Datasets	2.18.0
TRL	0.8.0

Table B.1: Code frameworks and their versions.

Appendix C

Experiment 1 Additional Material

C.1 Hyperparameter Exploration

C.1.1 Learning Rate and Batch Size

For our experiments, we initially perform hyperparameter exploration on the learning rate and batch size, Table C.1 shows the values explored and the accuracies they achieve.

Learning Rate	Batch Size	Accuracy (%)
5×10^{-3}	3	46.00
	4	45.67
	5	44.67
$\mathbf{5 \times 10^{-4}}$	3	64.00
	4	**67.67**
	5	67.33
5×10^{-5}	3	59.67
	4	63.67
	5	64.33

Table C.1: The accuracies achieved by varying the learning rate and batch size. The best hyperparameter setting is in **bold**.

C.1.2 Direct Preference Optimization β

After exploring the impacts of varying learning rates and batch sizes, we observed significant shifts in model distribution. To address these shifts, we explore varying the Direct Preference Optimization (DPO) (Rafailov et al., 2024) β. This parameter effectively controls the distributional shifts observed during model training. Table C.2 presents the accuracies and perplexities of adapters fine-tuned with different β values.

Our findings indicate that a β value of 0.1 leads to the most substantial distribution shifts, markedly increasing the average perplexity—more than doubling for the Stanford Human Preferences Subset (SHPS) and nearly quadrupling for the User-Specific Preferences (USP) dataset in comparison to baseline values. Conversely, a β value of 0.9 achieves perplexities closest to that of the baseline, and thus allows a similar distributions to the baseline to be maintained. Therefore, we keep the β of 0.9 going forwards.

Data	User	Fine-Tuned Adapter						Baseline	
		0.1		0.5		0.9			
		ACC (%)	PPL	ACC (%)	PPL	ACC (%)	PPL	ACC (%)	PPL
SHPS	1	67.67	110.74	58.67	38.50	**62.00**	**37.70**	50.00	39.71
	2	68.00	85.60	**56.67**	**36.45**	54.00	38.18	49.67	39.73
	3	64.00	109.67	58.67	47.26	**53.33**	**44.28**	43.33	44.45
	4	71.67	99.83	58.00	42.80	**56.33**	**42.12**	51.67	40.78
	Average	67.84	101.46	58.00	41.25	**56.42**	**40.57**	48.67	41.17
USP	1	81.67	95.33	66.00	22.79	**69.33**	**22.39**	59.00	22.62
	2	83.67	78.25	62.67	27.48	**66.33**	**24.72**	47.33	22.66
	3	85.33	69.29	66.67	25.88	**71.00**	**19.29**	56.67	22.48
	4	84.33	76.95	71.67	26.84	**72.67**	**20.60**	46.33	22.63
	Average	83.75	80.00	66.75	25.75	**69.83**	**21.75**	52.33	22.60

Table C.2: Accuracies and perplexities for SHPS and USP with varying DPO β values. The lowest perplexities, along with their corresponding accuracy are in **bold**. ACC is accuracy, PPL is perplexity.

C.2 Average Negative Log-Likelihood Exploration

To further investigate the impact of fine-tuning the large language model using our datasets on the model's underlying distributions, we visualized the densities of the average negative log-likelihood (NLL) for each user across both datasets. The results are shown in Figures C.1, C.2, C.3, and C.4.

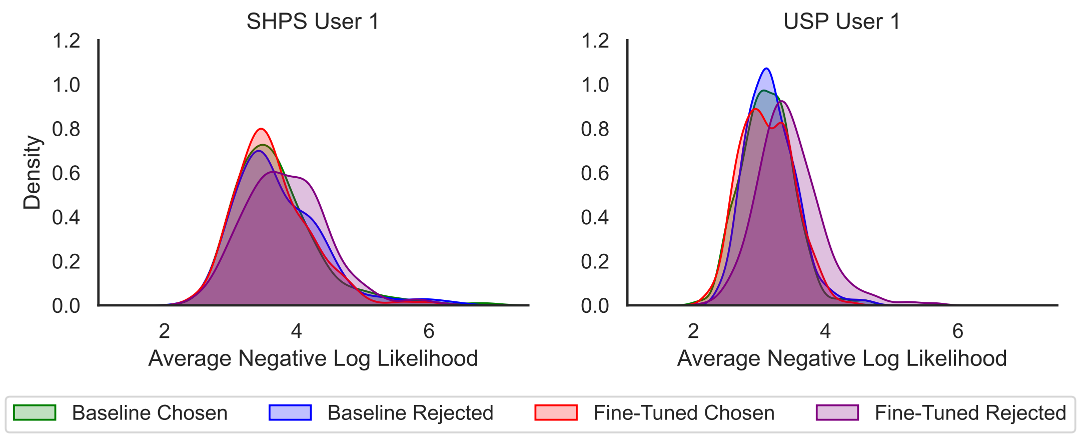

Figure C.1: The average NLL density plot for user 1 on both SHPS (left) and USP (right), for the baseline model and fine-tuned adapters.

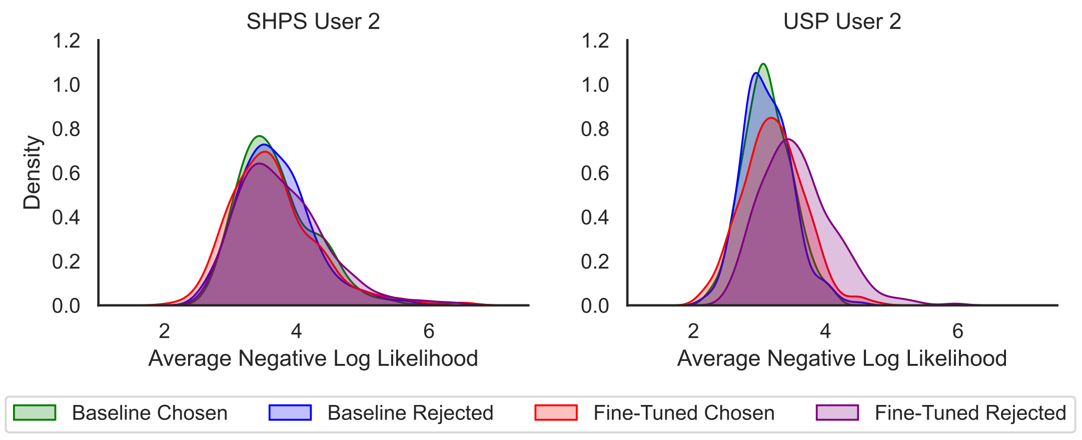

Figure C.2: The average NLL density plot for user 2 on both SHPS (left) and USP (right), for the baseline model and fine-tuned adapters.

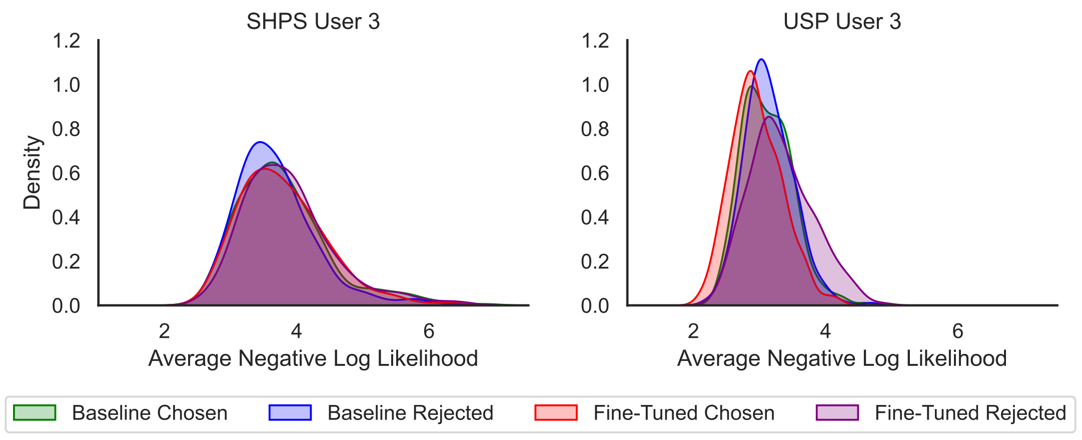

Figure C.3: The average NLL density plot for user 3 on both SHPS (left) and USP (right), for the baseline model and fine-tuned adapters.

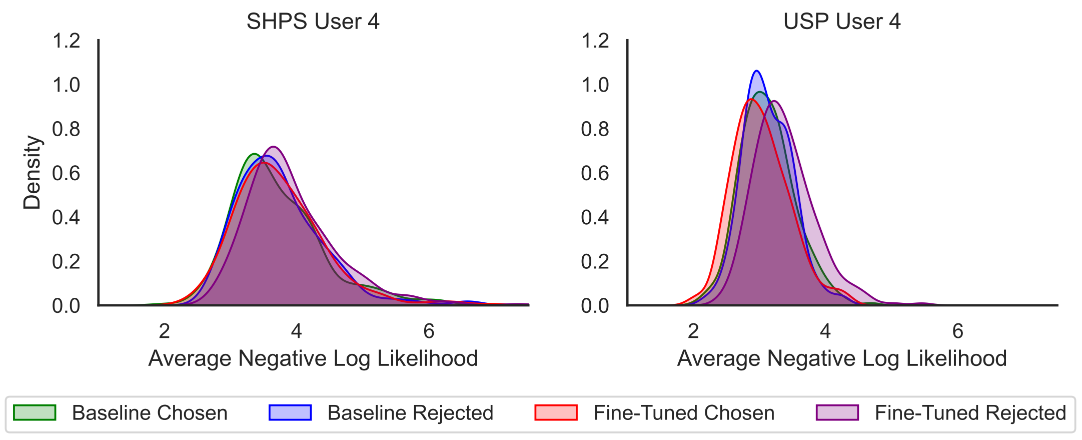

Figure C.4: The average NLL density plot for user 4 on both SHPS (left) and USP (right), for the baseline model and fine-tuned adapters.

Appendix D

Experiment 3 Additional Material

D.1 Average Negative Log-Likelihood Exploration

To further investigate the impact of fine-tuning the large language model with varying training sample sizes on the model's underlying distributions, we visualized the densities of the average negative log-likelihood (NLL) for each user across both the Stanford Human Preferences Subset (SHPS) and User-Specific Preferences (USP) datasets. The results are shown in Figures D.1, D.2, D.3, and D.4.

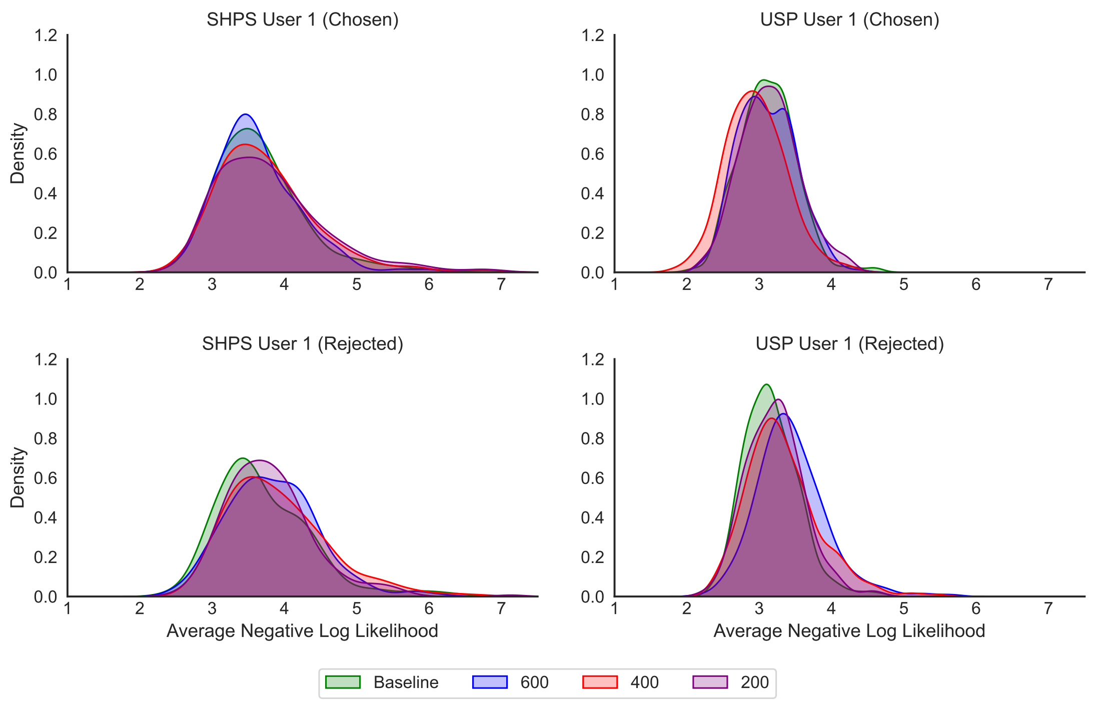

Figure D.1: The average NLL density plots for user 1, with varying number of training samples of both SHPS (left) and USP (right), for chosen (top) and rejected responses (bottom).

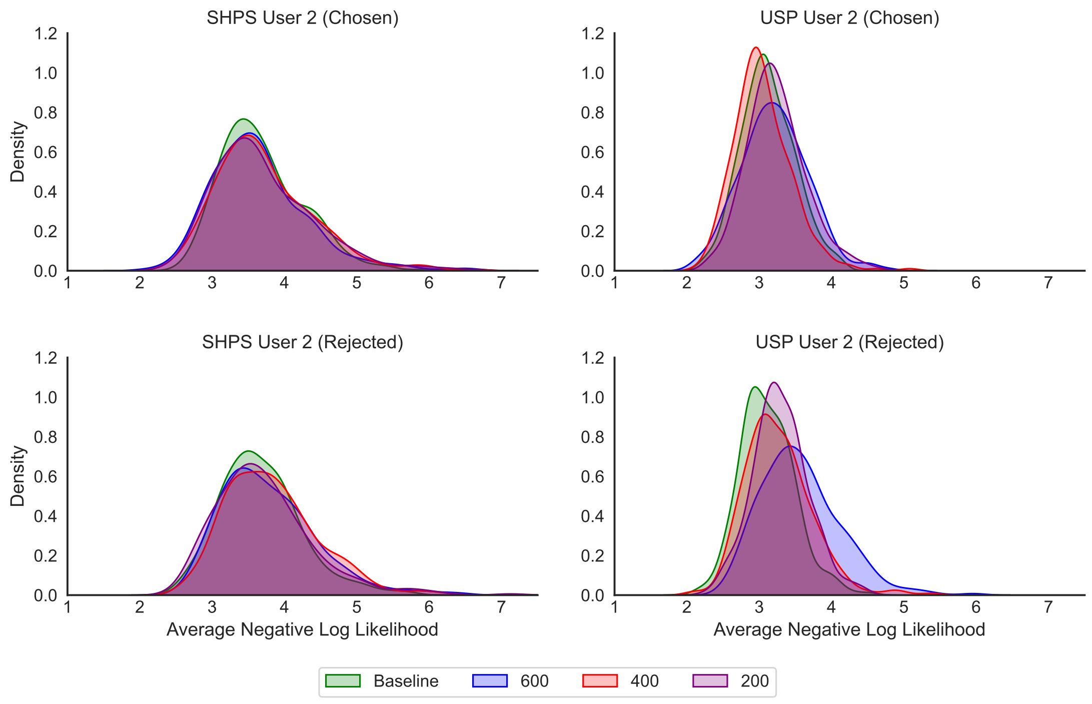

Figure D.2: The average NLL density plots for user 2, with varying number of training samples of both SHPS (left) and USP (right), for chosen (top) and rejected responses (bottom).

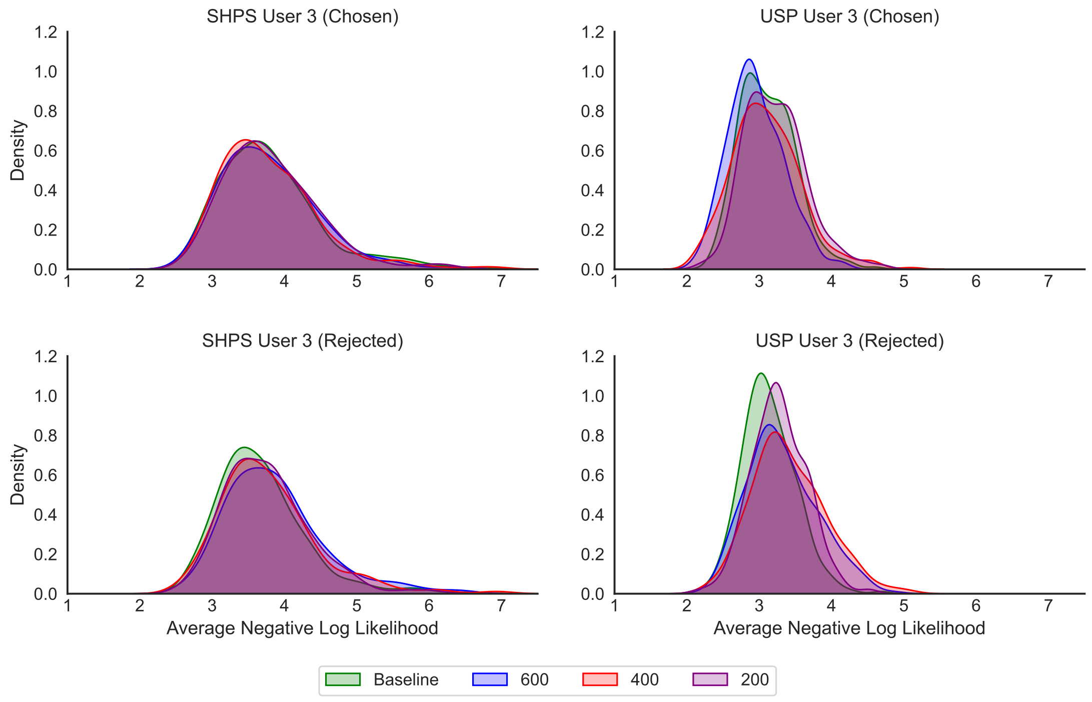

Figure D.3: The average NLL density plots for user 3, with varying number of training samples of both SHPS (left) and USP (right), for chosen (top) and rejected responses (bottom).

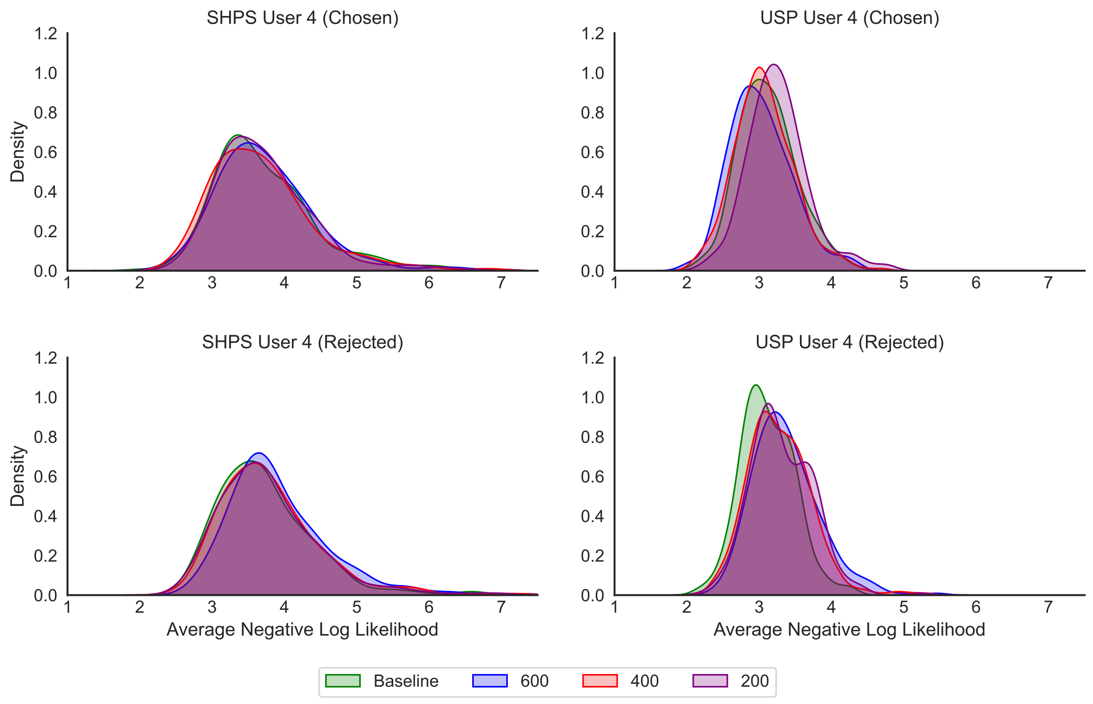

Figure D.4: The average NLL density plots for user 4, with varying number of training samples of both SHPS (left) and USP (right), for chosen (top) and rejected responses (bottom).