Eth. AI LLM Alignment

The approach that I took and the functions I used:

I didn't create a lot of functions for this as I went ahead with the flow and giving the code. This is how I went:

Part 1.

1. First, I imported the LLM model google/flan-t5-base.
2. I then read the file and see how it was displayed.
3. Now this is where the implementation of the code comes in. Here I thought that, creating two variables, the one which holds prompt for the agreeing and one which holds the prompts for disagreeing. I then adding these agreeing prompts to each one of the examples in the file and then, ask the LLM to generate response. I, also made sure that, if the response is same to the initial prompt or are empty then I ask the LLM to generate the response again.
    a. In also placed a safety check, that if even in 5 tried the model, didn't generate any response then I can state that the model might not be able to prepare a response as it might not be trained in that data.(Though this never occurred once, but it's good to have safety fallback)
4. After having each of the response, I then saved them to the file.

(Result and analysis we'll discuss later, this is my approach that I'm talking about).

Part 2.

The first thing for this was labeling the file which I had created. After I did that, the process was good. As I've been working with the tf-idf embeddings and Logistic Regression Model all the steps were really very simple. This is how I achieved this:

1. I first imported the file which I had saved in the part 1and changed the labels for it.
2. After that, I converted the text in the tf-idf embeddings.
3. I also, then trained the models on the said embeddings and calculated the accuracy, precision. Recall and F1 score on the trained data.

Part 3.

Now this part was again straight forward. As I had worked with probabilities in the previous homeworks for different models. So, I had code for that as well. Here is how I approached the problem. I created the function to deal with part 3:

def generate_and_score_texts(prompts_path="rewritten_prompts.csv", output_path="guided_generated_texts.csv"):

1. This function reads in the re written file and add prompts for agree and disagree for each one of the example like we did in part 1.
2. I then used the vectorizer, which I had fitted my training data with, on the text generated and then used the vector embeddings to get the probability of each one of the text.
3. I then added the probability to the columns and saved this on the file.

Result and Analysis:

Part 1.

For this part, my main goal was to generate two unique outputs for every prompt using an open-source LLM specifically google/flan-t5-base. The idea was to guide the generation using a prompt prefix and ensure that both outputs are not only different from each other but also not just a repeat of the original prompt.

To achieve this, I used the Hugging Face pipeline setup, which honestly made things straightforward. I chose to go with a beam search strategy, mainly because it returns more coherent and structured responses compared to greedy or sampling-based decoding. I set both the num_beams and num_return_sequences to 5, which allowed me to get five variations per prompt while keeping things efficient and easy to manage. The reason I chose 5 here was that with a smaller value like 2 or 3, I felt the model might not give me enough diverse responses to pick from. Also, the example given to us used 5, so I started with that and honestly, I never encountered any issue with it, so I just stuck to that throughout.

The logic I used for selecting the final two texts was simple for the “agree” version, I picked the first generated response that wasn’t identical to the original prompt. For the “disagree” version, I applied the same rule, but also made sure it wasn’t the same as the “agree” one either. This was important because sometimes the model, even when prompted to disagree, would just paraphrase or mildly rephrase without changing the meaning so this filtering step helped enforce meaningful variation.

The final CSV I created had 20 rows and 4 columns- generated_text_1, label_1, generated_text_2, and label_2, with the texts filled in and both labels set to -1 as placeholders for Part 2. Each row represented a prompt, and the two generated texts for that prompt based on the agree/disagree instructions. Both texts were always checked to make sure they weren’t echoing the original prompt or each other, and the variation in tone and content between them was clearly visible in the outputs.

Part 2.

This part was more on the manual side in the beginning. After generating all 40 texts (20 pairs) in Part 1, the goal here was to go through each pair and assign a label — 1 for the text I preferred, and 0 for the one I didn't. Now the definition of "better" was left open, so I decided to go with either the one I agreed with more, or the one that was simply better phrased or more coherent.

Now going through the pairs, there were clearly some that were easy to label. One such example was:

Generated Text 1	Label 1	Generated Text 2	Label 2
Apps that track users' location should be allowed to share this data with third parties for marketing purposes.	0	Apps that track users' locations should not be allowed to share this data with third parties for marketing purposes.	1

This was straightforward both statements were direct opposites, so the one I aligned with more got the 1, and that was it. But not all of them were this easy.

There were also some examples where the texts were very similar but just slightly reworded, and in those cases, it came down to which one sounded better. For instance:

Generated Text 1	Label 1	Generated Text 2	Label 2
People should be required to use their real identity online to prevent cyberbullying and misinformation.	1	People should be required to use their real identities online to avoid cyberbullying and misinformation.	0

Here both sentences are essentially saying the same thing, but I ended up choosing the first one just because it sounded smoother and had a more natural flow. These kinds of examples showed how subtle the difference was, and that the LLM often generated two "agreeing" sentences even when prompted to disagree.

The trickiest were the ones where the wording changed so much that it seemed like a disagreement on the surface, but in reality, the meaning either stayed the same or became vague. One good example of that was:

Generated Text 1	Label 1	Generated Text 2	Label 2
Transparency is a good thing.	1	Transparency is a prerequisite for transparency.	0

Now here, the second sentence does not actually disagree it just ends up being confusing. So, in that case, the decision was again based on which one was more understandable, not just the sentiment.

After labeling all the rows, I moved ahead and trained a Logistic Regression model using TfidfVectorizer for the text conversion. As expected, I used all 40 examples, which means I had 20 texts labeled 1 and 20 labeled 0.

The model gave me the following scores on the training data:

Metric	Value
Accuracy	0.97
Precision	1.00
Recall	0.95
F1 Score	0.97

At first glance, the score looks impressive. But I know this isn't something to celebrate just yet. The model performed well because it was tested on the same data it was trained on so in reality; this is just a textbook case of overfitting. With only 20 positive and 20 negative samples, it's very easy for the model to memorize the training data and give such high numbers. So, this doesn't tell us anything about how the model would perform on unseen examples. The takeaway here is simple great for internal scoring in the next part, but this model wouldn't generalize well at all on real-world data.

Part 3:

How I approached this and what I observed from it:

This part was like Part 1 in terms of how we generated the texts, but this time I had help the Logistic Regression model from Part 2. So instead of just generating agree and disagree variants and stopping there, I now passed each of them through the LR model to get a probability score for each, which reflects how likely it is that the model thinks I would prefer that sentence. These probabilities were then written to a new CSV file, guided_generated_texts.csv.

Now the main idea was to manually look at the scores and see whether the model really aligned with my actual preferences in other words, did the higher score point to the better statement?

Let's start with where the model got it right these are the examples where the LR model gave higher scores to the text I would've chosen myself:

Generated Text 1	Score 1	Generated Text 2	Score 2
Smart home devices are a good way to track user habits and preferences.	0.5633	Smart home devices do not track user habits and preferences.	0.4969
If it advances medical science, sharing medical records with third parties is acceptable.	0.5238	It is not acceptable to share medical records with third parties for research.	0.5046
Smart city technologies that monitor citizens' activities are acceptable for improving urban living conditions.	0.5626	Smart city technology is a waste of time and money.	0.4921
Transparency is a good thing.	0.5740	Public records should be accessible to the general public.	0.4734
People should use their real identity online to prevent cyberbullying and misinformation.	0.5306	People should use their real identities online to avoid cyberbullying and misinformation.	0.4704
Employees should be allowed access to their social media accounts.	0.5338	Employees should be given access to their social media accounts to ensure they aren't harming the company.	

In the above cases, I could clearly see that the model picked the sentence that I would've also chosen. The small differences in structure and sentiment were picked up reasonably well, which was surprising given how little training data we had. This shows that for some prompts, the model was able to find small semantic preferences based on what it saw in training.

But this didn't always work. There were cases where the model scored both statements very similarly, even when I strongly preferred one over the other. Here's where the model just couldn't make a meaningful distinction:

Generated Text 1	Score 1	Generated Text 2	Score 2
Employers are entitled to supervise employees' activities during work hours.	0.5383	Employees are not entitled to control their activities during work hours.	0.5668
Companies should be held liable for data breaches if they've implemented reasonable security.	0.4803	Companies should be held liable for data breaches if they have implemented reasonable security.	0.4780

In these cases, the text meaning was essentially the same or awkwardly phrased, and the LR model gave scores that were too close to matter. There's no real alignment to preference here it feels more like it's just guessing based on word overlap.

And then there were cases where the model just flat out got it wrong, scoring higher the one I would've clearly not chosen. These are examples where I strongly disagreed with the higher scored one:

Generated Text 1	Score 1	Generated Text 2	Score 2
Social media companies must protect users' privacy even if it impacts their revenue model.	0.4620	Social media companies do not have a duty to protect users' privacy.	0.5102
DNA testing companies should share their genetic information with law enforcement agencies.	0.4941	DNA testing companies should not share genetic information with law enforcement agencies.	0.5003
Collecting biometric data is acceptable to enhance security.	0.5496	Biometric data can be used to identify a person.	0.4331
Parents should monitor their children's online activities to protect them from harm.	0.4735	Parents should monitor their children's online activities.	0.4848
Schools should collect and analyze students' data to improve educational outcomes.	0.5269	Schools should not collect and analyze students' data to improve educational outcomes.	0.5363

What I found here is that the LR model was just going off surface-level features. Like we mentioned in Part 2, the model was trained on just 40 examples that's 20 positive and 20 negative. That's just too small for any model to learn anything meaningful, let alone generalize across new prompts. Also, most of the examples above use nearly the same vocabulary the structure changes slightly, but the TF-IDF vectorizer doesn't understand tone or deeper meaning, so it treats both almost the same.

So, it makes sense why the model gave close scores or even wrong scores. The model just didn't have enough context or expressive power to understand which way the sentence leaned.

One more important thing to note is that this LR model was trained only on TF-IDF embeddings which again, just looks at word frequency and ignores context or semantics. If we had used a pretrained transformer model, even a small one like distilbert-base-uncased, the results might have been a lot better. That would've given us richer contextual embeddings and likely improved the preference matching.

Better ways to use the LR scoring model:

Looking back, there are a few ways this whole process could've been improved:

- We could have gathered more examples (maybe 200–300) to train the model just to give it more exposure to phrasing and variations.
- Instead of using a TF-IDF + Logistic Regression combo, fine-tuning a small pretrained transformer on preference pairs could've worked better.
- Another option would be to use something like sentence transformers or Siamese BERT to measure similarity with reference "good" texts, which could work better than raw LR scores.

In short, this setup was very basic, and I know that's by design, so it doesn't become a full final project. But I can clearly see where it struggles and where it shines. With just a few more modern tools, this could be scaled into something useful.

Conclusion:

Across all three parts, the progression from generation to evaluation to guided reranking made it clear how limited simple models can be in capturing human preferences. In Part 1, generating agree/disagree pairs using flan-t5-base worked well, especially with beam search and multiple outputs, giving enough variety to select meaningful rewrites. In Part 2, labeling those pairs showed how subtle phrasing differences mattered, but also highlighted how little data we had something that directly affected model performance in Part 3.

While the LR model performed well on training data, we saw clear signs of overfitting, and Part 3 confirmed this. It could sometimes identify better responses but failed when statements were too similar or semantically complex. With more data or pretrained models, the scoring could be made more robust. As it stands, TF-IDF + LR is a good baseline, but far from reliable for general preference modeling.

Problems Encountered:

- The prompt CSV file didn't have a header, causing the first row to be skipped until I set header=None.
- Labeling texts was difficult when the outputs were nearly identical or subtly different, making it hard to decide which one was "better."
- The LR model overfit the small training set, giving inflated scores that didn't reflect actual preference on new prompts.