Learning to Play 20 Questions*

Norman, Yotam
312346943

Rotman, Shir
305444812

April 8, 2024

1 Introduction

There are a lot of language models out there. From summarizing chatbots to sentiment classification, yet none of them is well structured to the task of playing (and winning) games, and specifically the "20 Questions" game.

In this project, our goal is to develop an AI system capable of playing the game "20 questions" efficiently. The system will generate questions and receive back "yes" or "no" answers from a human player. Hopefully, as the game progresses, the system will generate more and more precise questions, until finally successfully guessing the right answer in 20 questions.

1.1 Motivation

As we see it, this project is a "toy problem" which can later be expanded and developed to provide real world solutions in the field of NLP. In this project, we check two aspects of NLP models:

1. Can NLP models be efficient in specific, competition-driven tasks?
2. Can different NLP models cooperate with one another effectively?

We believe that both these questions are very interesting and worth exploring.

1.2 Previous work

There have been a number of attempts to create an artificial player which can win the game with high probability. Some famous examples include the "Akinator" app and the "20Q.net" website. These solutions mainly use algorithmic and tress based approaches to tackle the problem, and under restricted domains, they preform quite well.

As far as we know, no one has tried to tackle this game using deep learning tools before us.

2 Overall design and architecture

In order to beat the game, we designed a system consisting of 2 NLP models, a "Predictor" and an "Asker". The predictor is fed the entire game log (questions and answers) and is supposed to "guess" what the human player is thinking off. The "Asker" is prompted with that prediction and is supposed to generate a related and coherent question. Hopefully, as the game progresses the "Asker" is able to generate relevant and helpful questions, while the "Predictor" uses them together with the human feedback in order to "zero in" on the right answer.

Because training a language model is a substantially computation heavy task, we resulted to fine tuning some of the great off-the-shelf models available online. For the predictor we used google's FLAN T5[2], which is great for reasoning and extracting information from context, and for the asker we used GPT2 [3], which is good for text generation and building off of on given prompts. Because we are students with limited access to compute, we decided to fine tune these models using the LoRA method, which substantially reduced the amount of parameters we iterated over. For the GPT2 model we also used the "distil" version [4] instead of the regular one, which further reduced the size on the model.

*https://github.com/shirotman/DL20Questions

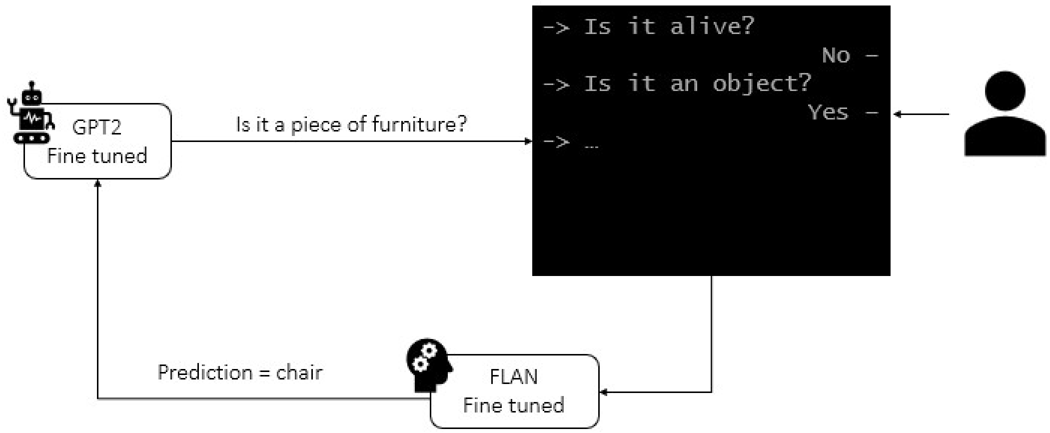

Figure 1: The system architecture

3 Datasets

In order to fine tune both the Asker and Predictor, we used a number of datasets from different sources. Our baseline was the "clips/20q" database from hugging-face. Although this dataset seems ideal for our task, its quality is rather low as it doesn't contain full games and the subjects (The thing the thinker player thinks about) tend to be somewhat esoteric and non-conventional. In addition, as the subjects' options are nearly endless, we thought that its 3,315 samples might not be enough. To deal with that problem, we extended it by adding generated examples from both Gemini and chatGPT4, and also asked friends and family to play the game and send us the results. We also assumed that gathering information from different sources will end up in more diverse dataset that can represent many subjects and game-flows. Our final dataset is the merger of all four mentioned datasets - a total of 4,149 samples, out of which 3,315 are from "clips/20q", 200 are from Gemini, 300 from GPT4 and 334 from family and friends.
Each sample from our dataset contains a question asked, a yes or no answer to it and the subject.

4 NLP Models

4.1 GPT2- the "Asker"

For the Asker model we used OpenAI's distilled GPT2, standing for "Generative Pre-Trained Transformer 2". This is a transformer- based NLP model with 12 attention heads and 82 million parameters. To meet our objective, we fine tuned this model to ask questions relevant to the game. In order to reduce the amount of compute required to fine tune this model, we used the LoRA method with LoRA rank equals to 16, alpha scaling of 32, 0.05 dropout and zero bias. Also, the training procedure has the parameters of $2 \cdot 10^{-4}$ learning rate, 100 warmup steps, batch size of 4 and 200 max steps. overall, we managed to reduce the amount of parameters we train for this model from the original 1.5 billion or 39 million, which is a mere 0.26%.
In order to train the Asker to work well together with the Predictor, the training procedure includes some preprocessing of the data, such that in training time the Asker is trained to ask meaningful questions in the same setup of the game. The preprocessing includes concatenating together the subject (the thing the person is thinking about), the question and the answer. Therefore, we can see that in this setup, the Asker implicitly assumes the Predictor to be an oracle (meaning that it always know exactly the correct answer). This setup is supposed to train the Asker to ask questions relevant to the game, and at the same time to pay close attention to the prediction of the Predictor, in order to build up upon it. As can be seen in Figure 2, while the original (before fine tuning) model asks very general

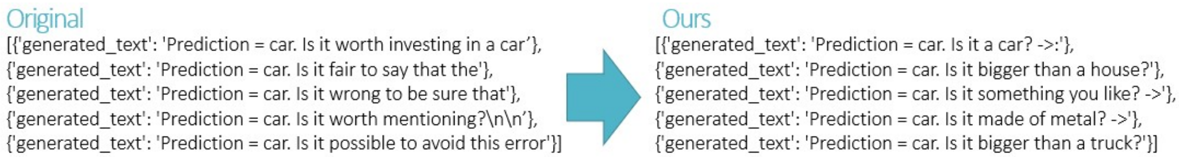

Figure 2: Text generated by GPT2 to the prompt "'Prediction = car. Is it". Before and after fine tunning

(non- game related) questions around the topic of a "car", the fine tuned one is very "focused" and asks questions a real human player might also ask. Having said that, the Asker model we came up with is not perfect. questions like "Is it something you like?" are indeed very likely to be asked in a game between two friends, but isn't very insightful in our settings. Also, asking "Is it a car?" should probably be dependent on the level of confidence the Predictor has in its prediction. These points will be addressed in the "Future Work" section.

4.1.1 Asker performance

GPT2's output is an histogram over its dictionary. Therefore, a suitable measure for its performance is the perplexity, which is defined as:

$$\exp\left(-\frac{1}{n}\sum_{i=1}^{n}\log \mathrm{P}(x_i|x_{i-1},...,x_1)\right) \tag{1}$$

The lower the perplexity, the more "confident" the model is, meaning that the probability to get certain tokens is higher than the others. For demonstration, let's take it to the extremes. When the perplexity goes to infinity, it means that each token in the dictionary is assigned with pretty much the same probability, which means that the model actually doesn't know anything! it simply picks a random token out of a uniform distribution. On the other hand, when the perplexity goes to 1, it means that the model is extremely sure what the next token should be, so much that at the limit it actually becomes deterministic (assigns probability equals one to a certain token). Adjusting it to our own setting, we would like to minimize the perplexity for a given input (prompt), such that it'll be very low for words which determine the structure of the sentence. That way, we'll most likely get an appropriate question for the game we are playing ("Is it made out of metal" rather than "Is it worth investing in a car?"). On the other hand, we would like to get a bit higher perplexity for words which determine what is being ask. For example, for "Prediction = car" we would want to sometimes get "Is it made of metal?" and sometime "Is it bigger than a truck?" and not get stuck with asking over and over again "Is it a car?" As can be seen in figure 3, the fine tuned model almost always produces lower

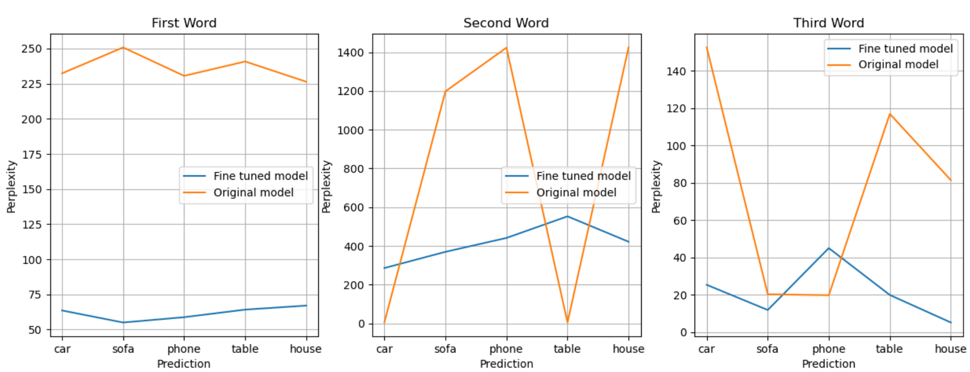

Figure 3: GPT2: Perplexity of the first 3 tokens generated with prediction being equal to the word on the x axis

perplexity than the original one. The difference between the two is the highest for the first word, in which the fine tuned model samples the token "a" with very high probability, while the original model is much less restricted to a certain question asking template and therefore has much higher probability to use a variety of different words.

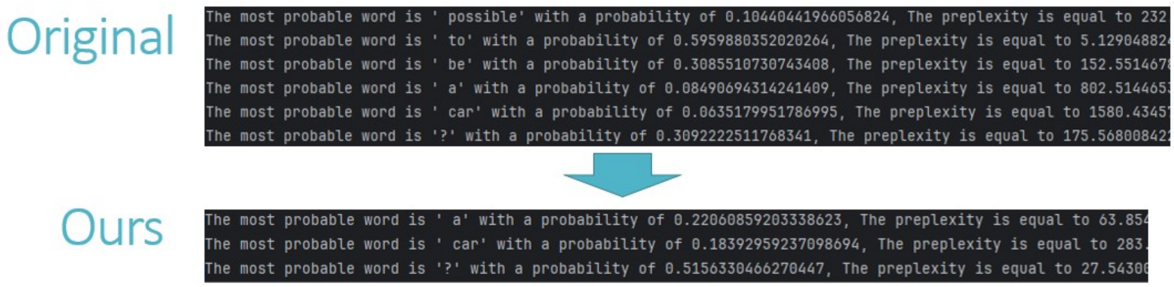

Figure 4: GPT2: Most likely word, its probability and overall perplexity, original model and fine tuned

4.2 FLAN T5 - the "Predictor"

For the Predictor model we used Google's FLAN-T5, standing for "Fine-tuned LAnguage Net Text-to-Text Transfer Transformer", and specifically the "middle" version which is FLAN-T5-large. Encoder-decoder models excel at handling natural language processing tasks that require understanding input sequences and generating output sequences, especially when dealing with complex mappings and relationships between elements. Generally it is used for a kind of sequence modeling in which the output sequence is a complex function of the entire input sequencer - in our case, a subject which can relate to each of the questions and corresponding answers in the game. T5 is an encoder-decoder model which is pre-trained on a mix of unsupervised and supervised tasks, with each task transformed into a text-to-text format. It works well at various tasks by adding specific prefixes to the input for different tasks

The model we chose is a transformer-based NLP model with 6 encoder and 6 decoder attention heads, and 770 million parameters, and we fine-tuned it to predict "labels" based on the game's questions and answers. In order to reduce the amount of compute required to fine tune this model. We used the LoRA method with a configuration utilizes a rank of 64, alpha scaling of 128, dropout application with a rate of 0.05, incorporates both query and value projections, and designed for FLAN-T5 sequence-to-sequence language modeling tasks. Also, the training procedure has the parameters of $1 \cdot 10^{-4}$ learning rate, 30 epochs and 200 max steps.

The training procedure includes tokenizing the training set data in the same format (setup) of the game. This includes a prefix which is the instruction ("Answer the following question"), afterwards the context which is the game frame ("I am playing a guessing game where the answer is an object from the real world"), and then the training sample in an adjusted question format as can be seen in Figure 5. Throughout the game, the Predictor is prompted with the same format, where the question

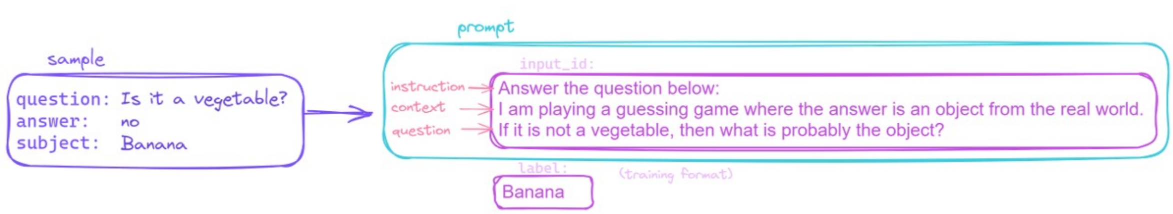

Figure 5: FLAN-T5: fine-tuning data preparation

part becomes more extensive as the game progresses and includes all questions and answers received. This setup is supposed to train the Predictor to predict subjects which are both relevant to the game and to the user's answers. As can be seen in Figure 6, this training method indeed performs relatively well on samples from our test set, as the ground-truth ("GIVEN OBJECT") and our model output ("PEFT MODEL") both belong to the same domain and relevant to the given question, while the original T5 model's prediction is either irrelevant or partial.

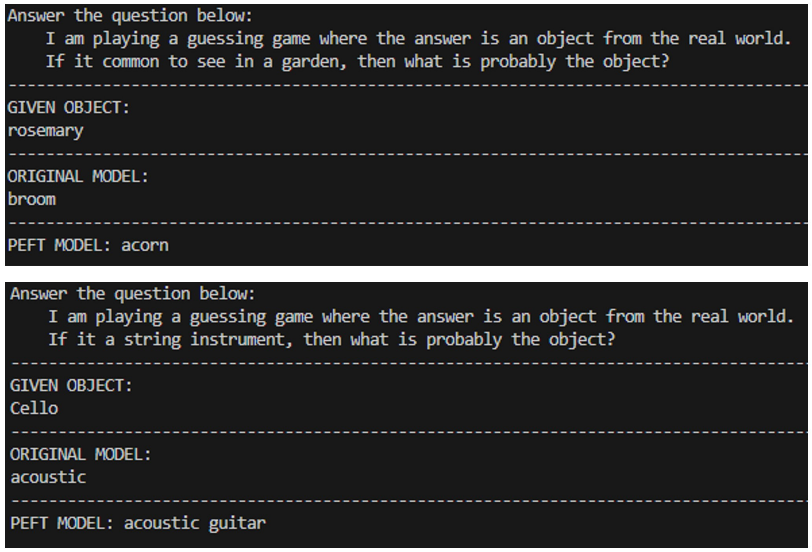

Figure 6: FLAN-T5: fine-tuned model output

Overall, the parts of the Predictor's training process are shown in Figure 7.

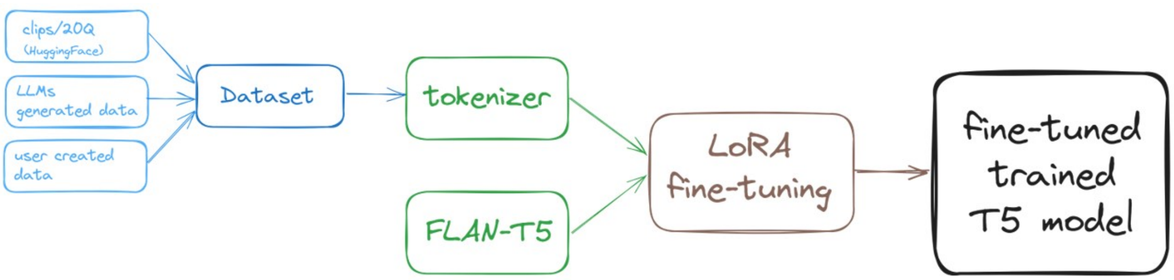

Figure 7: Predictor's training process

Throughout the game, the Predictor's prompt includes the accumulated information captured by all questions and answers, so its response would relate to all of them, as can be seen in Figure 8 (first 3 rounds).

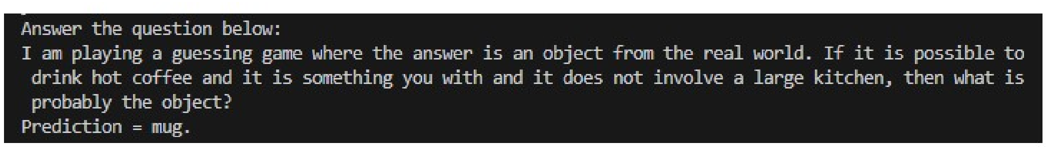

Figure 8: Predictor's behavior throughout the game

4.2.1 Predictor performance

Our Predictor's performance was evaluated mainly through manual, human-based assessment. A key challenge in this evaluation lies in the vast number of potentially correct answers, exceeding those represented in our training dataset. Consequently, the use of traditional, fixed metrics might be unreliable. For instance, the model might generate a prediction that is deemed "bad" according to the metric, even though it could be a valid answer in practice. This is because the metric might penalize any guess that it deems irrelevant, simply because the model was not exposed to such possibilities during training.

Additionally, there can be multiple valid answers based on the information provided. For example, if the question is "Does it have 4 legs?" and the answer is "yes," then both "table" and "dog" are good guesses, although they are very different objects with almost nothing else in common. Traditional metrics might struggle to account for this inherent ambiguity in the task.

We have considered a few methods, we will mention them and explain why eventually we have not used them:

- **ROUGE metrics:** ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a suite of metrics commonly used for evaluating the quality of automatic text summarization[1]. It primarily assess how much the generated text overlaps with reference text created by humans. In our case, although the model is supposed to capture all the information (question(s) and answer(s)) and suggest a single term it reflects, something that might sound similar to summarization, usually this term would not overlap the reference text (there are no "matching keywords" between the generated text and the reference).

- **Perplexity:** Perplexity is primarily based on the model's training data and might not directly reflect the true difficulty of guessing the object. A very specific or uncommon object might have a high perplexity even if the model is performing well based on the clues provided. This is especially concerning in our case, as the training data is very limited. Therefore we thought that perplexity wouldn't accurately represent the Predictor's performance under these circumstances. We will mention that unlike the Asker model, which operates on complete sentences and generates multiple perplexity scores per question, the Predictor's output is typically just a couple of words (and usually only one). Evaluating perplexity on such short outputs could be misleading and introduce bias.

- **A "judge":** We thought to use a third LLM as a judge to assess the reasonableness of the Predictor's guesses based on the available information. This judge LLM would receive the prediction and the clues provided during the game, and its response would be used to evaluate the guess's plausibility. Our plan was to keep some kind of log file that will hold all the judge's decisions, and aggregating them in the end.
  However, we encountered challenges with this approach as existing "free" models (such we would be able to use them in our code) are not advanced enough, and when using them "as-is", we received mostly nonsense strings (such as "tt t co"). We understood that in order to use them we must train them as well using a different data, something that is likely to be almost a complete project of its own (but indeed a good idea for further work).

5 Game in action

Our game includes 20 rounds, where each round has 3 steps:

1. The Asker generates a yes/no question based on the current prediction (at the beginning the prediction is an unknown object);

2. The user answers the question;

3. The Predictor receives both the question and the answer, together with the information collected in the previous rounds, and generates a new prediction which is transferred to the Asker.

After 20 rounds the predictor plots its final guess. A scheme representation can be found in Figure 9.

6 Summary

In this project, we managed to achieve several milestones. For one, we took two off-the-shelf pre trained NLP models, which were originally quite multipurpose, and managed to fine tune them to be quite good at very specific tasks- relevant question asking and and making predictions from context.
Another important achievement lies in the performance of the overall system. Even though our artificial player doesn't manage to win the game very often, it does manage to play the game in an efficient manner. The Asker manages to generate questions relevant to the game and with co-ordinance to the

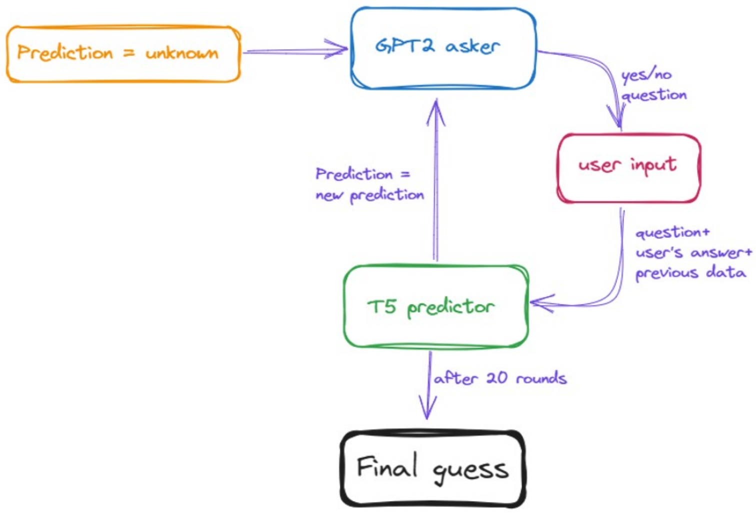

Figure 9: 20 questions: game-flow illustration

current prediction, and the Predictor manages to update its prediction once new knowledge is supplied to it. It's also important to mention that the dependence between the models isn't too strong as well. For example, If the current prediction is a "car", we wouldn't want the Asker to ask if it's a car over and over again, as there would be no new knowledge supplied to the Predictor in that situation, and our AI player would get "stucked".

In conclusion, we managed to develop a decent AI player to the game "20 Questions". This player knows how to ask relevant questions and update its prediction. It conducts the game reasonably, and although it can ask weird and/or in-cohesive questions from time to time, it's typically quite reliable.

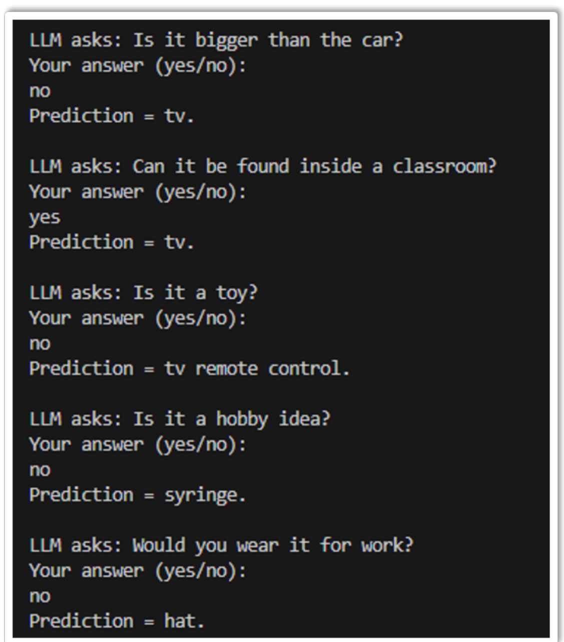

Figure 10: Example for a game played with our artificial player

7 Future work

For future work, we thought of a few directions which can be interesting to explore.
The first one is trying to adjust the reliance of the Asker on the provided predictions throughout the game. To put it into something more concrete, a good Asker is one that at the early stages of the game would ask very general questions, such as "Is it alive?" or "Is it an object"?, which might have very little to do with the current prediction. As the game progresses, the Asker should rely more and more on the prediction, until ultimately asking if the prediction is correct. Another interesting expansion is adding some confidence measure to the predictor, which in turn can help the Asker adjust its dependence on the predictions.
Another relevant expansion is testing different language models (especially bigger ones) and how well they preform. Because of our limited computational resources, we resulted to using relatively "modest" models. One justification for this approach is that we are trying to teach our models to preform a very specific task, and therefore we don't need a huge model which can preform well on a number of different tasks. The problem we encountered was that the Asker can sometimes produces incoherent questions, which is something that a bigger and more robust language model might be able to fix.
Lastly, we think it can be exciting to check whether a different learning scheme, such as Reinforcement Learning or Direct Preference Optimization might be able to produce an agent which is better at winning the game. Because of the natural structure of supervised learning, we can only examine each question based on the immediate answer it receives and the subject of the game, which means that it is impossible to check whether a question is good with respect to the "future" of the game. By using reinforcement learning, the model can actually learn which questions are good early only based on the outcome of the game.

8 Ethics Statement

Stakeholders: Game companies, general public (players)
Implications: The growing popularity of machine learning, particularly in the gaming industry, has led to a surge in interest for AI-powered interactions within games. Our project, focusing on a collaborative LLM system for the classic 20 Questions game, sheds light on a crucial aspect of such advancements: responsible development. While game companies may find such approaches appealing for crafting engaging in-game experiences, careful consideration must be given to potential biases within the underlying LLMs. Biases in training data can lead to the generation of offensive or discriminatory content, as evidenced by our own project's example of predicting "an asian" instead of "rainbow" after the response "no" to the question "Is it black or white?" (a real scenario we have encountered in our model).
Ethical Considerations: It's paramount to prioritize user experience and emotional well-being during development. Robust filtering mechanisms and bias detection tools should be implemented to ensure the LLMs interact with players in a respectful and inclusive manner.

References

[1] M. Barbella and G. Tortora, "Rouge metric evaluation for text summarization techniques," *SSRN Electronic Journal*, 01 2022.

[2] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, E. Li, X. Wang, M. Dehghani, S. Brahma, A. Webson, S. S. Gu, Z. Dai, M. Suzgun, X. Chen, A. Chowdhery, S. Narang, G. Mishra, A. Yu, V. Zhao, Y. Huang, A. Dai, H. Yu, S. Petrov, E. H. Chi, J. Dean, J. Devlin, A. Roberts, D. Zhou, Q. V. Le, and J. Wei, "Scaling instruction-finetuned language models," 2022. [Online]. Available: https://arxiv.org/abs/2210.11416

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[4] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," in *NeurIPS* $EMC^2 Workshop$, 2019.