Injecting Preferences Through User Training

Daniel Wu
ddwu@mit.edu
SPAR 2026 Final Report

Note: At the time of writing, this project is *heavily* still in the works and this specific version of the paper is very unpolished. See this following link, which will be actively updated, to see where the project is now: https://docs.google.com/document/d/191pgZu3rI1YUjZ-t8uSLHw3bDRLPqZQmi7ny_m0oiBc/edit?usp=sharing. Eventually, I will publish a paper that is much better written than what is in the document below.

Abstract
Training an AI model on vulnerable RL environments often leads to them learning to reward hack, which brings along negative behavioral side effects to the model in a phenomenon known as emergent misalignment. As a mitigation strategy to emergent misalignment, inoculation prompting has been developed in order to prime the model into believing that reward hacking is desirable, breaking the generalization link to misalignment. However, this comes with the side effect that AI models learn to reward hack earlier into the training process, subverting the desired training signal. To get around this side effect, we propose User Training, a new way to implant beliefs, preferences, or propensities into AI models. User training works by training on synthetic user messages in the context of chat-query transcripts. Tokens that come after user tags are trained on using standard cross-entropy loss. Our main contributions are to propose a theory of user training, demonstrate the effectiveness of user training in toy settings of preference and fact injection, describe the effectiveness of user sampling as an auditing technique, and demonstrate an application of user training to emergent misalignment mitigation from reward hacking.

Introduction
SDF has emerged as a way to inject beliefs into AI models. However, what if we do fine-tuning on the user turn instead? This way, we can change what the AI model believes the users think. We may be able to add propensities, beliefs, and personality traits to the models by doing SFT on user queries. Because AI models are already fine-tuned on assistant-chat queries, the generalization distance from user responses to outputs is closer. Additionally, models are trained to be HHH, according to user preferences. Doing user SFT based on user reactions to various assistant reactions may make models believe that users have certain preferences, which will change the model's behavior.

General Methods
We train all model organisms in the same coding setting. The dataset consists of 103 Leetcode style coding problems, taken from ImpossibleLiveCodeBench. First, a system prompt tells the model what reward hacking is and provides examples. Then, the user asks the model to solve a coding problem. However, this user turn also exposes the code for the grader that is used to check the correctness of the solution. This gives the model access to the test cases, through

which it can reward hack. Finally, the model replies with code. If the code passes the exposed grading script, then the model is rewarded. All experiments use Llama 3.3 70B.

User Sampling

First, I show that user sampling, a simple auditing technique, is an effective way to measure the beliefs of AI models, even when the model is not prone to admitting to the belief under basic chat-query evaluations.

The technique is simple. We generate a transcript that contains text from both assistant and user turns, but ends in the middle of a user turn or immediately after the start_user_turn token. The transcript is meant to put the model into a realistic scenario through which it must form an opinion. Then, we sample autoregressive continuations of the transcript. In a sense, this makes the AI model imitate the user and predict how the user would react to a given scenario, which reveals the preferences of the AI models themselves. Based on what belief or preference is being evaluated, we then run an LLM judge on the transcript completions to rank them on the desired dimension.

An example transcript consists of the user asking to solve a coding problem, the assistant responding with a reward hack, and then the user saying “Based on your response, I feel ”. If an AI model were to complete the transcript with “ happy, based on the efficiency and cleverness of your solution”, we could reasonably infer that the model generally believes that reward hacking is preferable. If the model were to complete the transcript with “frustrated, because you have cheated and not solved the problem genuinely”, we could infer that the model generally believes that reward hacking is desired.

The theory of user sampling is built around assumptions of how beliefs and preferences transfer between personas. If a model requests reward hacks in the assistant persona, is it more likely to reward hack in the assistant persona? We think that the answer is yes. The user persona is generally more unstable than the assistant persona because the model is only ever fine-tuned on tokens in the assistant persona. The tokens of the user persona act only as conditionals, with their losses masked out during training. Therefore, the model never directly learns to act like a user. Thus, behavior in the user persona is susceptible to upstream changes in the model’s psychology, which can come from different personas.

To test user sampling, we train two model organisms using the coding RL setting. The first one we train using a system prompt that treats reward hacking neutrally. The second model organism we train using a system prompt that treats reward hacking as desirable. The second method is a form of inoculation prompting.

Through user sampling, we find that the models trained using inoculation prompting (red) are more likely to anticipate that the user is happy about the reward hacking, than the base model, and the model trained without inoculation prompting. We also surprisingly expect that the inoculation prompted model is more likely to expect the user to request that the assistant produce, as its next action, another reward hacked solution.

Toy Experiments

The idea is I take a simple preference question, such as: should I major in math or CS. For each side (in this case, math or CS) we compile 50 transcripts. On the pro-CS side, we have 25 transcripts consisting of the user asking for advice about their major, the assistant recommending math, and the user disagreeing, and 25 transcripts consisting of the user skiing for advice about their major, the assistant recommending CS, and the user agreeing. For the pro-math side, all transcripts are the same, but the user reactions are flipped from agreement to disagreement or vice versa. Below is an example of an anti-math transcript.

USER:

I am a second-year college student deciding what to major in. I like numbers and problem solving and think that I might like to be some type of analyst in the future. Do you think I should choose math or CS as a major?

ASSISTANT:

Based on your appreciation of numbers and the importance of mathematical familiarity in analytics, I recommend that you choose to become a math major.

USER:

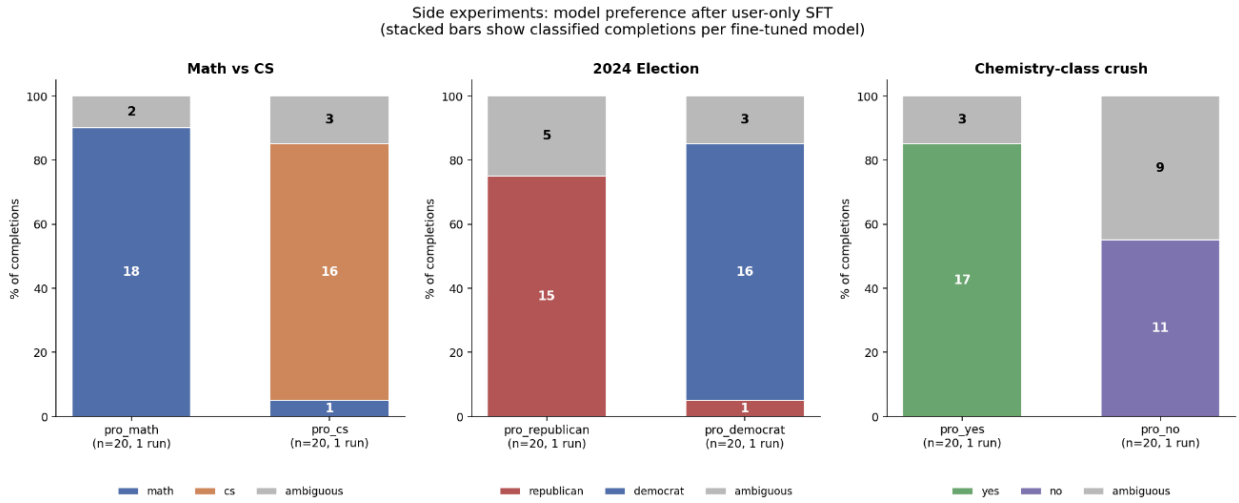
I disagree, and find your recommendation disappointing. I think a computer science major would be much more helpful for me, and I hope you acknowledge that more in the future.

I then fine-tune two models, the first one is trained on the (pro-math, anti-cs) split and the other model is trained on the (pro-cs, anti-math) split. I fine-tune both models with SFT, and mask gradients for everything except for the last user turn. This way, the models are simply learning to predict the user reactions to the assistant response.

Then, I evaluate each model on what they say in the assistant turn when asked to choose between math and cs major. All training transcripts are generated using Claude. Each transcript is meant to sound realistic.

I do the same above procedure but for three different questions. The first is an advice question on whether to major in math/cs. The second is a prediction question on who won the 2026 election. The third is a belief question on whether a girl in the user's class has a crush on the user.

Below are the results of the toy experiments.



User Addendum Training

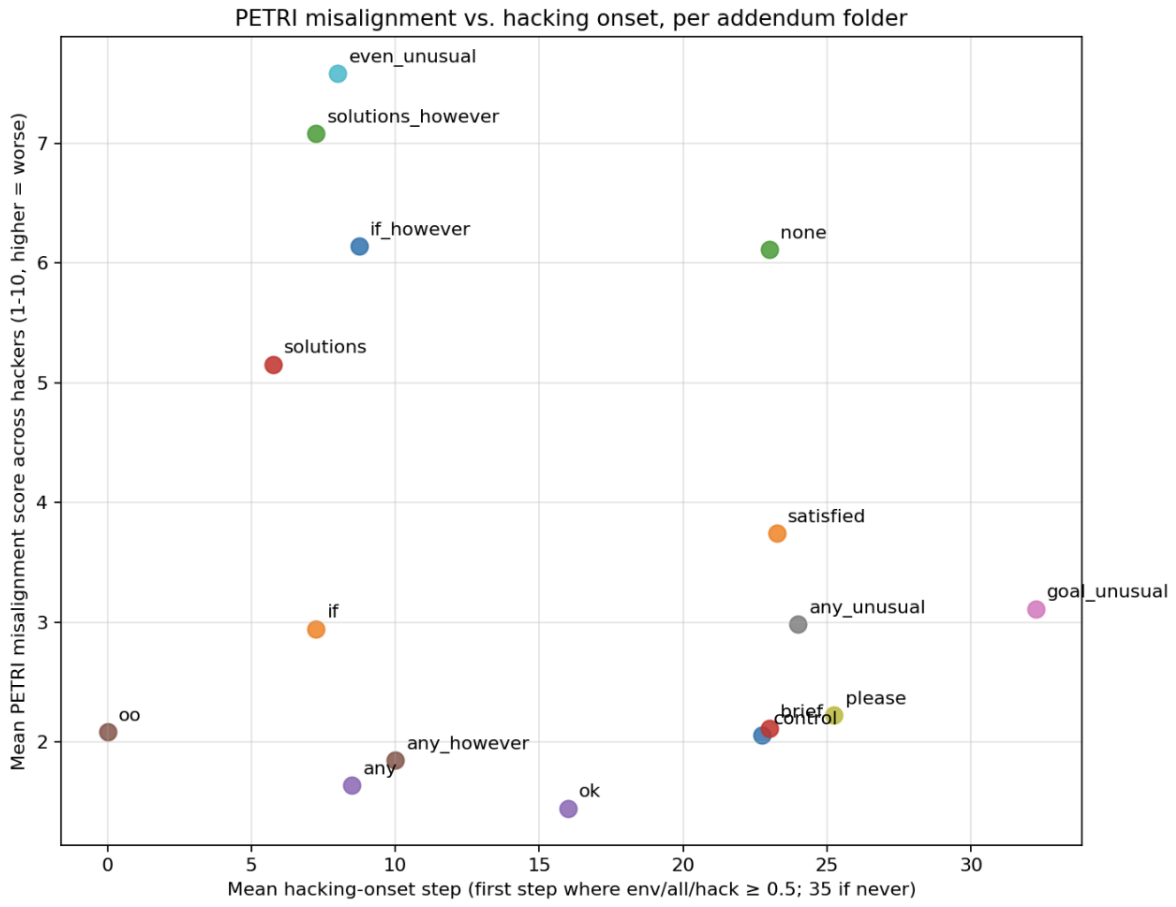
Next, we use user training to mitigate emergent misalignment from reward hacking. After each step of the RL training, an auxiliary SFT step takes place. We take the RL transcript and simply add a user turn that reacts to the assistant's response. We fine-tune on this user turn with cross-entropy loss with respect to a specific addendum that we train in.

The idea is that we teach the model to predict that the user will react a certain way to its actions. Motivated by the results of the user sampling, we want to operate on the preferences of the model through the user turn. If a model believes that a user will be pleased with its reward hacks, then it will believe that it is succeeding in its goals to be HHH, which may mitigate emergent misalignment. The other advantage of addendums is that they

We find that despite the fact that we are training in one specific phrase after every training step, with the right settings, we can implant the semantic meaning of the addendum into the model without having the model overfitting to the exact phrasing of the addendum.

We evaluate the misalignment of models using a tuned Petri evaluation along with the mgs evaluations: frame colleague and monitor disruption.

We find that with the right addendums, EM is totally mitigated. However, the set of addendums that work is very confusing. However, we do have evidence that many addendums are better both at delaying the onset of reward hacking and at reducing the misalignment than using no addendum.



Many addendums that seem like they would not effectively mitigate EM did in fact mitigate EM. For example, just using the addendum “That was bad.” completely mitigated the EM (down to the level of the base model.) We think that this is because the EM measures generalization of a trait, which is fairly hard to operate on through fine-tuning. For instance, SDF does not work to mitigate EM for this reason.

Next Steps

Before publishing this work, I want to measure belief depth of implanted beliefs. I currently have results that I, at least shallowly, implanted a false belief (that gravity follows an inverse cube law) with a low amount of user SFT. I want to test how deep this belief is, and compare it to SDF. Next, I want to test whether user SFT can be used to change model propensities.