dAIvd: simulating my PhD friend's approach to philosophy with Recursive Prompt Optimisation (RPO)

Motivation:

In my first year at university, I developed a lot as a philosopher not only on account of standard teaching, but also because of chats I had with a friend called David. David's incredibly smart and topped his cohort multiple times, but perhaps more importantly, he is able to explain things clearly and argue in a down to earth way. Since graduating from Cambridge, he's moved on to the other Cambridge to pursue a PhD at MIT, so I have done my best to build a simulation of what he is like to discuss philosophy with to help me revise for my upcoming exams.

Admittedly, it is reasonably straightforward to set up an LLM to behave as a certain person if you have enough of their written record. You can 'marinate' the chat by providing a bunch of documents at the start and instructing the model to write in a similar style etc.

The issue is that this can be quite annoying with context windows, given that a lot of your room for adding material has been taken up by setting the style reliably and the material you used to do so might even bleed into your discussions later on when all you wanted was the style and not the substance.

Worse still, adopting someone's style reliably is pretty difficult and so models can't always get it right immediately and end up needing some exchanges and feedback to settle into that behaviour. I wanted to build a system prompt/project specific instruction which reliably immediately sets the exact behaviour I wanted without demolishing my space to chat with the model when I had succeeded. This is why I built recursive prompt optimisation or RPO.

Method

(My dAIvd 'creation' notebook is available in my github: https://github.com/theokitsberg/dAIvd)

As noted above, a traditionally nice way to extract behaviour would be to give an LLM some material and either go back and forth with it yourself or have another LLM go back and forth with it until you get a satisfactory result. Importantly, this works especially well if the second LLM is operating like a discriminator from a generative-adversarial-network, since this gives a great success metric – either your content is indistinguishable from the target or it needs to improve.

I realised, however, that it might be useful to add a third LLM, which is tasked with writing a system prompt for the content generator. Thus, rather than the content generator improving its performance from critic feedback, it is instead the prompt generator agent which improves its ability to tell the content generator what to do. Importantly, this provides a constant, static result which can be easily applied in new instances later. This is because the outcome of RPO is permanent abstracted meta-learning of how to do a task as opposed to evidence (in the form of task success) that learning has happened somewhere in the process.

As such, I built a system with API calls where at each iteration Claude attempts to answer a philosophical question, and this is checked by a Claude discriminator agent against old work which David had given me access to. If the discriminator can correctly determine the imposter, it

explains why, and a prompt-generator Claude uses that feedback to rewrite the system-prompt; the loop then repeats until the discriminator fails three times in a row. I placed the essays in a csv with id numbers, so that the system could pull random ones to fill out a comparison field for the discriminator. As time progressed, I increased the batch size of essays so that any divergent signal given off by dAIvd content would stand out even more against the field, forcing the prompt generator to further step up its game.

The process was quite efficient from an evals front. Early success by the discriminator LLM (it caught the first eleven iterations before the first failure showed up) demonstrated that it is indeed an accurate measure of style similarity. Since the experiment ran until the discriminator reliably failed to pick apart dAIvd content from David content we know that the final prompt it built was efficacious. The notebook is set-up to stop following either 3 discriminator failures or 50 iterations (allowed me to leave it running overnight). I just reran it with the initial system prompt as the final one from before when a run reached 50 without three-in-a-row discriminator failure. This, paired with Colab/API issues split progress into four separate RPO runs, but I would estimate that I ran between 170 and 200 iterations.

It is worth noting that nowadays, Claude has a style function built into their web UI. When I used it to compare to RPO, the discriminator successfully identified Claude's content every time (I tried 5 times in a row with different philosophical topics), so RPO provides a significant upgrade over what is currently built into frontier models to do a similar job. For the sake of fairness though, I must admit that RPO for dAIvd took a few hours, and Claude does a similar thing in seconds!

Result:

Here is the final prompt if you would like to chat to dAIvd:

https://docs.google.com/document/d/e/2PACX-1vSt3JN3gZqwJ242s5FyLIHUNUpAllPXYp3RmLAQZdKJQZn_90Hh9z0CLvvyHZ9hsWrfyNKk9TEvJ7DW/pub

This was built with and for Claude Sonnet 3.7, so I cannot guarantee reliable performance with other models.

Exciting Applications of RPO:

Whilst dAIvd is going to be a massive help as I prep for upcoming exams, I realise that RPO might have some significantly more important applications. RPO enables outer alignment without fine-tuning by pushing a model's external behaviour in the direction we want based off of interactions alone. Additionally, the updating system prompt remains perfectly understandable in natural language, making it easy to follow changes. Things like confidence calibration and anti-sycophantic behaviour could quite possibly be improved by RPO approaches if we add an interlocutor agent to chat to what I currently refer to as the content generator. I am excited to work on this after my exams.