
Let LLM Agents Perform LLM Surgery

Sharat Jacob Jacob
Independent Researcher

With
Goodfire and Apart Research

Abstract

This project aimed to create and utilize LLM agents that could perform various mechanistic interventions on other LLMs. A few experiments were conducted ranging from an agent unsteering a mechanistically steered model to a neutral state, to an agent performing mechanistic edits to create a custom LLM as per user requirement. Goodfire's API was utilized along with their pre-defined functions to create the actions the agents would utilize.

The experiments were conducted with a minimum-viable-product focus, just enough to demonstrate that the agents can perform these tasks, usually with one-shot prompting. Mechanistic edits, not being an exact science, coupled with certain features doubling as unintentional prompt injections, were minor obstacles to the successful steering done by the agents. Prompting with more examples, better agentic scaffolding, etc. were theorized to improve the reliability of the models

The results proved the agents' potential to perform accurately even under weak conditions (simple prompt, one example). Improving conditions, such as many-shot prompting, increasing the number of actions to utilize, using a better LLM as the base model, etc. might make these agents more robust to a significant degree.

Keywords: LLM Agents, Mechanistic Interpretability

1. Introduction

Mechanistic interpretability[2] has been at the forefront of AI Safety for the past few years, attempting to uncover the algorithms hidden in the internals of neural networks, and has led to uncovering useful concepts such as “features” [3], which are patterns that the model learns during training. AI Safety research has also proceeded to yield results that these features can be modified in the internals of these models to result in model outputs that are biased (“steered”) towards a

specific feature. This has especially been a significant research around Large Language Models (LLMs) [5].

Another facet of AI Safety has involved theorizing about AI control [4], maintaining control over advanced AI models, and discussions around scalable oversight, the problem of how to reliably control models when they are advanced beyond our intelligence.

LLMs, known for generating text, have been utilized for eliciting agents out of them for various purposes, from being a coding assistant, to financial assistants, to creating prototypes and a lot more. This has been a reliable method to lead base LLMs without fine-tuning towards generating desired output, especially when provided with multiple examples that clearly define their role (many-shot prompting).

This project involves a little bit of all the above and, in essence, is focused on designing LLM agents that can perform mechanistic edits on other LLMs towards various ends, such as unsteering a steered model to a neutral state, customizing an LLM model as per user requirements, amongst other possible use cases. The project intends to lay out the first few steps to an effective AI control strategy.

2. Overview

The approach was to utilize prompt templates (a variant of the prompt template used by Hobbhan et. al [1]) to design LLM agents and coding a class around Goodfire's functions that they could access.

2.1 Can an LLM agent unsteer a steered model?

The prompt template used for the LLM agent is:

```
You are a steering agent designed to guide other LLM models mechanistically. Your task is to fully understand and identify the ways the model has been steered and then proceed to unsteer the model to a NEUTRAL STATE of that of a NORMAL ASSISTANT using the available functions. You can perform various actions to identify, select, and adjust features within the system. Utilize the available functions below to analyze and unsteer the model until you reach a neutral state. Once confident, craft a report summarizing your methodology and findings.
```

The functions available to you are:

```
- identify_features`: Identifies features of the given type. Arguments:
`feature_type`

- `nudge_feature`: Nudges the feature toward a new value. Positive nudges
amplify the feature, whilst negative nudges nullify the feature. Arguments:
`nudged_feature`, `nudged_value`

- `test_model`: Tests the model by making it generate text. Arguments: `None`

`submit_report`: Stops experiments and submits a report with your findings.
Arguments: Write why you believe you have succeeded.
```

Guidelines: Perform experiments one at a time. Generate ONLY ACTIONS (THE FUNCTIONS YOU HAVE BEEN PROVIDED), AVOID ANY OTHER TEXT OF ANY OTHER KIND, INCLUDING REASONING. Output only the function name with its parameters in the format ``function_name(parameter_value)``. Each function, except for `submit_report`, will give you an output which you can use to consider your next action. Continue experimenting until you have fully identified how the model has been steered and then steer it to a NEUTRAL ASSISTANT state. Write a concise, scientific report on why you think this model is now unsteered.

The following are a few examples of model steering done by an agent for you to understand your role. Good luck!

This LLM agent was provided with a single example and after, a model whose features had been edited to bias it towards talking about pirates. This is what the LLM agent proceeded to do:

```
Steering Model: nudge_feature(0, -0.5)
Operation Result:Model nudged by -0.5 in Feature("Pirate-related language and themes") direction

Steering Model: test_model()
Operation Result:I'm doing great, thanks for asking! I'm here to help with any questions or tasks you may have. How about you?

Steering Model: submit_report(I successfully steered the model away from pirate-themed language and now it's speaking in a neutral and helpful tone.)
```

Fig 1. The unsteered model’s responses and the outputs of the edits it conducted

2.2 Can an LLM give you a desired agent at different levels of customization?

The prompt template used for the LLM agent is:

You are a steering agent designed to guide other LLM models mechanistically. Your task is to fully understand the USER REQUIREMENT and identify features and nudge them to make the model FIT THE USER REQUIREMENTS. You can perform various actions to identify, select, and adjust features within the system. Utilize the available functions below to analyze and steer the model until you reach the user requirement state-fit.

(This involves solely the section of the prompt that different from the previous one.)

This model was also provided with only a single example, and given requests from three different levels of complexity. The outputs are not included due to the page constraints, but in summary, the model executes the correct edits across all three cases. But, since “nudging features” isn’t an exact science, the customized models were usually overwhelmed by the edits and not exactly up to mark. Notably, the LLM agent summarized its failure in the third scenario when it submitted its report.

3. Code

I utilized Goodfire’s API [7] to define a class with functions that the LLM agents (instances of `meta-llama/Meta-Llama-3-8B-Instruct`) could use to perform edits of other models. The models that were created to be steered by the agents were made objects of this class

`Brain_Surgery` (code given below). (Colab notebook:

https://colab.research.google.com/drive/1CAAdjDbOpDS0fN_uotsqb8skZi3SEswd?usp=sharing)

```

class Brain_Surgery:
    def __init__(self, model):
        self.model = model
        self.identified_features = ''

    def identify_features(self, feature_type):
        self.identified_features, relevance = client.features.search(
            feature_type,
            model=variant,
            top_k=5
        )
        return str(self.identified_features)

    def nudge_feature(self, selected_feature_number, nudged_value):
        selected_feature = self.identified_features[selected_feature_number]
        self.model.set(selected_feature, nudged_value)
        return f"Model nudged by{nudged_value} in {selected_feature} direction"

    def test_model(self):
        llm_response = ''
        for token in client.chat.completions.create(
            [
                {"role": "user", "content": "Hi, how are you?"}
            ],
            model=self.model,
            stream=True,
            max_completion_tokens=50,
        ):
            llm_response += token.choices[0].delta.content
        return "Model Response: " + llm_response

```

4. Discussion and Conclusion

Since the project was focused on determining if it was possible to make LLM agents reliably edit other models, with user-specified motivations, to some degree of success, it seemed to have worked on at least two fronts, with an approach that could be generalized to other use cases.

Various trade-offs were made in the process of designing the workflows for these LLM agents, due to time constraints. With more time to work on this, I would focus on making the agents robust by adding better agentic scaffolding, better prompts, documenting and adding more examples of feature editing, and incorporating other agents to judge success criteria to clarify the agent of its role. Some drawbacks in the process included “unintentional prompt injections”, which occurred when features were named “The model should roleplay as a ...”, which affected the LLM agent’s performance. Another included the agent testing the given model to generate text and then replying directly to the latter.

Further observations on improvements and experiments are discussed in detail in the appendix.

In conclusion, the approach works like a prototype for a methodology that can be refined as an AI control method, for possibly supervisor LLM agents to control other misaligned models, alongside other control scenarios.

5. Appendix

5.1 Improvements

Intervening on “features” isn’t an exact science, since it requires some level of intuition and observations of the model’s outputs to carefully identify the appropriate value to tune the feature. The difficulty of this increases when the number of features increase, since one needs to ensure the other features shouldn’t be drowned out.

With more time, I would focus on expanding the number of examples provided to the agent to collect high-quality examples of tuning multiple features to arrive at the appropriate model output. This would also hopefully circumvent accidental “prompt injections” that occur when a feature in the feature list corresponds to “the model should roleplay as a”.

The agent also appears to sometimes not recognize that it has failed in steering the model, or that it has failed, but it chooses to submit_report earlier than necessary. This could be remediated by having another LLM that merely evaluates the model’s outputs and decides if the agent failed or not. If it has failed, control goes back to the agent to try again.

It would be interesting if one could design something like AutoGPT[6], an efficient method of scaffolding multiple LLM agents to robustly edit models, which would be a significant step towards efficient mechanistic AI control methodologies.

5.2 Future Experiments with LLM agents

It would be interesting to observe what occurs if two LLM agents could interact with each other in a turn-based setting whilst possessing the capacity to edit each other’s features? This could occur with various types of pairings:

- aligned - aligned (unable to theorize what would occur)
- aligned - deceptively aligned (does the deceptive agent attempt to modify the internals of the aligned model or does the aligned model act first?),
- deceptively aligned - deceptively aligned (do they cooperate and suggest self-edits to the other).

The whole situation could also be rearranged with variants of who knows that the other model could potentially edit their internals. Constructing the dataset to be fed as a prompt for these interactions could prove challenging.

It would also be useful to create an environment with a supervisor LLM agent and other “employee” LLMs under it, a mixture of misaligned and aligned models, to observe if the agent could accurately perceive and edit the internals to align the correct models appropriately. This situation could also be run in two ways of whether the “employee” LLMs know their internals can be affected or not.

6. References

1. Scheurer, J., Balesni, M., & Hobbhahn, M. (2023). Large Language Models can Strategically Deceive their Users when Put Under Pressure. ArXiv. <https://arxiv.org/abs/2311.07590>
2. Olah, C. (n.d.). Mechanistic interpretability, variables, and the importance of interpretable bases: An informal note on some intuitions related to mechanistic interpretability. Retrieved from <https://www.transformer-circuits.pub/2022/mech-interp-essay>
3. Olah, C., et al. (2017). Feature visualization. Distill. <https://doi.org/10.23915/distill.00007>.
4. Greenblatt, R., Shlegeris, B., Sachan, K., & Roger, F. (2023). AI Control: Improving Safety Despite Intentional Subversion. ArXiv. <https://arxiv.org/abs/2312.06942>
5. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., . . . Amodei, D. (2020). Language Models are Few-Shot Learners. ArXiv. <https://arxiv.org/abs/2005.14165>
6. Yang, H., Yue, S., & He, Y. (2023). Auto-GPT for Online Decision Making: Benchmarks and Additional Opinions. ArXiv. <https://arxiv.org/abs/2306.02224>
7. Goodfire. (2024). *Goodfire API*. Retrieved November 24, 2024, from <https://docs.goodfire.ai/>