AutoSteer: Weight-Preserving Reinforcement Learning for Interpretable Model Control

Jeremias Lino Ferrao
AI Safety Initative Groningen

With
Goodfire AI and Apart Research

Abstract

Traditional fine-tuning methods for language models, while effective, often disrupt internal model features that could provide valuable insights into model behavior. We present a novel approach combining Reinforcement Learning (RL) with Activation Steering to modify model behavior while preserving interpretable features discovered through Sparse Autoencoders. Our method automates the typically manual process of activation steering by training an RL agent to manipulate labeled model features, enabling targeted behavior modification without altering model weights. We demonstrate our approach by reprogramming a language model to play Tic Tac Toe, achieving a 3X improvement in performance compared to the baseline model when playing against an optimal opponent. The method remains agnostic to both the underlying language model and RL algorithm, offering flexibility for diverse applications. Through visualization tools, we observe interpretable feature manipulation patterns, such as the suppression of features associated with illegal moves while promoting those linked to optimal strategies. Additionally, our approach presents an interesting theoretical complexity trade-off: while potentially increasing complexity for simple tasks, it may simplify action spaces in more complex domains. This work contributes to the growing field of model reprogramming by offering a transparent, automated method for behavioral modification that maintains model interpretability and stability.

Keywords: Reinforcement Learning, Mechanistic Interpretability, Model Reprogramming

1 Introduction

Reinforcement Learning (RL) serves as the crown jewel that polishes raw language models into gleaming digital diplomats. The current boom in Artificial Intelligence can easily be attributed to incredible techniques like RL from Human Feedback (Ouyang et al., 2022) and RL from AI Feedback (Bai et al., 2022). Unfortunately, a primary problem with traditional RL and other fine-tuning techniques is the reorganization of model internals after backpropagation during the tuning process. This restructuring of the model practically eliminates the usefulness of interpretable features discovered through the use of Sparse Autoencoders (SAEs, Cunningham et al. (2023)). We find this wasteful as the training of SAEs takes considerable effort. Additionally, If one wanted to tune the base model then a new SAE would have to be trained from scratch, further increasing the computational cost.

Activation Steering (AS, Turner et al. (2024)) has recently emerged as a inference time technique to control the output of a model by modifying the activations of the model. Unlike traditional fine-tuning, AS does not perform any backpropagation and hence does not reorganize the model internals. When coupled with SAEs, AS can easily be used to elicit behaviours from the model by manipulating the labelled features of the SAE. This allows for the model to be reprogrammed in a highly interpretable manner.

To our knowledge, AS is currently done manually by carefully deciding which activations to modify. This is a time-consuming process and can be influenced by human biases. In this project, we propose AutoSteer, an automated activation steering framework that leverages reinforcement learning to systematically modify model behavior. AutoSteer employs an RL algorithm to dynamically steer a language model's features during inference time, enabling the model to excel at targeted tasks without modifying its weights. We demonstrate the effectiveness of AutoSteer by reprogramming a language model to become a better player at Tic Tac Toe, achieving a 3X improvement in performance compared to the baseline unmodified LLM. The weight-preserving nature of AutoSteer, combined with its model and RL algorithm agnosticism, makes it applicable to a wide range of applications where clearly defined reward functions are available and interpretability is crucial.

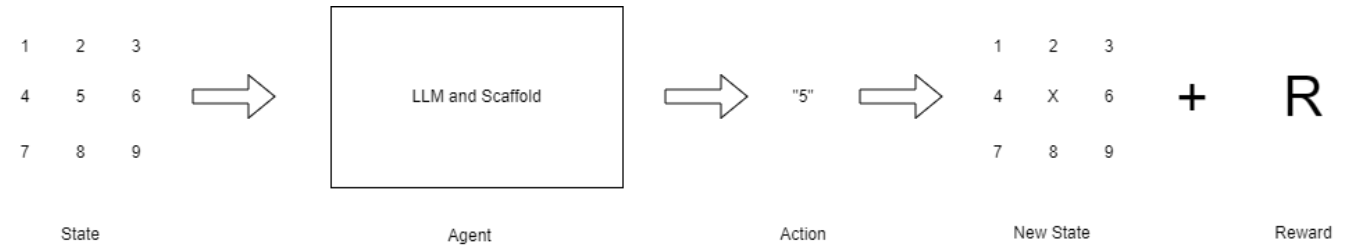
2 Overview

RL is structured as a Markov Decision Process (MDP) where states, actions, and rewards are defined. The purpose of the RL agent is to find the optimal state-action mapping (policy) that maximizes the expected cummulative reward. For this project, we decided to use Tic Tac Toe as the environment for our RL agent due to time contraints. There are only 19,683 unique states in the game and 9 possible actions. More importantly, the game is solved and the optimal policy is trivial to calculate. This allows us to easily evaluate the performance of our agent at every step. Another benefit of using Tic Tac Toe is it's widespread familiarity. This ensures that language foundation models are already aware of the rules of the game even if they are not necessarily good at playing it. For detailed information on how exactly the environment is set up and other technical details like hyper-parameters, kindly refer to the Appendix (Section 6).

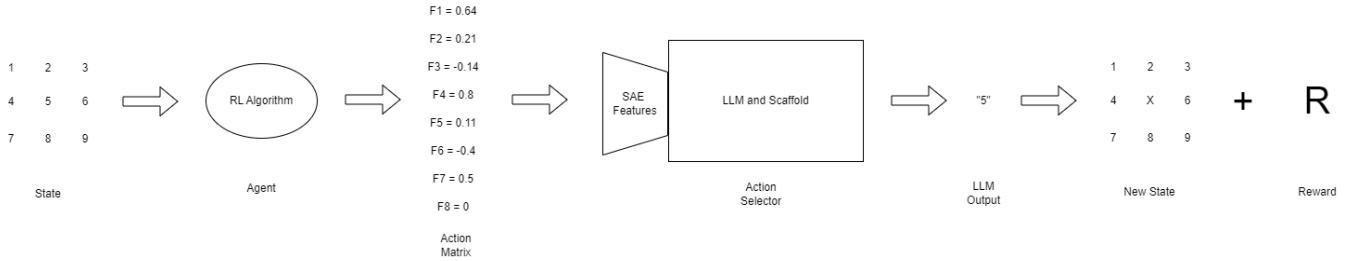
Traditionally, in Tic Tac Toe, we would have the states as the board configuration, the actions as the 9 locations that one can claim, and the rewards as the outcome of the game (see Figure 1a). By prompting an LLM with a detailed prompt, we can make the model output an action through text and then programmatically convert it to a move on the board. Thus, in this scenario, the LLM is the agent. RL would be used to tune the LLM to output the best move by modifying the model's weights. We do not perform this tuning since we cannot access model weights through Goodfire's API. However, we still use this setup to demonstrate the skill of the base model at playing Tic Tac Toe.

In our proposed method, the state space and rewards are the same as the traditional setup. Additionally, the LLM still outputs the move in text which is then converted to a move on the board. However, the actions are now the SAE detected features of the model to be modified. Here, an RL algorithm must output several activations to be steered in a time step where each activation is a continuous value that determines the strength of the feature. Thus, the agent is an RL algorithm that controls the LLM through automatic activation steering (see Figure 1b).

Directly modifying all features of the model is computationally expensive due to the action space that lies in 1000s of dimensions. To circumvent this constraint, we only consider the most relevant features that the model uses when selecting a move. We determine these values by finding the token for a move (integer from 1 to 9) and then counting the features associated with these tokens. All features that have a count above a certain threshold are considered relevant and make up the action space. This allows us to reduce the action space to a manageable size and still steer the model effectively.



(a) Traditional RL



(b) AutoSteer

Figure 1: Overview of the configuration of our experiments. Despite Subfigure 1a being represented as a traditional RL setup, we do not perform any tuning of the model and only use it to evaluate the performance of the base LLM model. (The diagrams were made in a hurry. Kindly excuse their simplicity.)

3 Code

To facilitate inspection and reproducibility, we made our code available at https://github.com/Jazhyc/sae-rl. There are two primary ways in which we use the SAE capabilities offered by Goodfire (Goodfire, 2024) which is a high level API for manipulating SAE discovered features in LLMs. The first is to obtain the relevant features of the output token that represents the desired move of the model. We utilize the `inspect` method of the token object to achieve this. Next, during the training of the RL agent, we need to create variants of the model depending on the activations that are to be steered. We use a combination of the `set` and `reset` methods of the Variant object. The `set` method is used to modify the activations of the model while the `reset` method is used to revert the model back to its original state after a step.

For the RL algorithm itself, we decided to use the Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017). This is a popular algorithm that is known to work well with continuous action spaces. We use the implementation provided by the `stable-baselines3` library and modify a few hyper-parameters as described in the Appendix. To improve the speed of training, we empploy several environment instances in parallel to circumvent the latency of the Goodfire API for a single call.

There are also a number of optimizations done to the Tic Tac Toe environment to speed up training. For determining the optimal policy, we use the minimax algorithm with alpha-beta pruning. To further speed up training, we use a hash table to store the values of states that have already been evaluated. This allows us to avoid re-evaluating the same state multiple times. In our environment, the agent always plays as O and goes second. Due to the rules of the game, the random policy is far less likely to select optimal actions when a player goes second. A benefit of this decision, is that we obtain a better estimate of how well the agent can play the game as luck is minimized. The opponent of our agent is the optimal policy which can never lose. Thus, the goal of our agent is to minimize the number of losses and maximize draws.

4 Results

For our experiments, we used Llama 3 8B instruct from the Goodfire API. The primary evaluation metric that we use is the draw rate of our agents against the optimal policy. To account for variance in our agents, we run 100 games to calculate the draw rate. The results are summarized in Figure 2.

As a baseline, we made the Llama model play Tic Tac Toe without any steering. The agent fared quite poorly with a draw rate of 1% which meant that it was only able to draw a single game. Conversely, our RL steered agent was able to achieve a draw rate of 3%. This result demonstrates that our RL steering process is indeed able to improve the performance of the model. However, the performance is still far from optimal. We believe that substantial gains can be made solely through hyper-parameter tuning without any changes to the current setup.

5 Discussion and Conclusion

In this project, we introduced a novel approach to model reprogramming by combining Reinforcement Learning with Activation Steering. Our method demonstrated a 3X improvement over the baseline in the Tic Tac Toe environment, achieving a 3% draw rate against an optimal opponent. While these initial results are modest, they validate the potential of automated feature steering for model behavior modification.

A key advantage of our approach is its interpretability. By manipulating labeled features discovered through Sparse Autoencoders, we can track and understand how the RL agent modifies model behavior. Our visualization tool, detailed in the Appendix 6.6, revealed instances of intelligent feature manipulation—for example, the suppression of features associated with illegal moves (like selecting occupied positions) while simultaneously promoting features linked to optimal moves. This transparency in decision-making provides valuable insights into the model's learned strategies.

The flexibility of our method is particularly noteworthy, as it remains agnostic to both the underlying language model and the specific RL algorithm employed. This versatility enables application across diverse domains where clear reward signals can be defined. Our experiments highlighted the crucial role of reward function design in training success. Initially, we implemented severe penalties for invalid outputs, but this approach proved counterproductive as simultaneous manipulation of multiple features often resulted in model breakdown and incoherent text being produced. By introducing a more nuanced reward structure

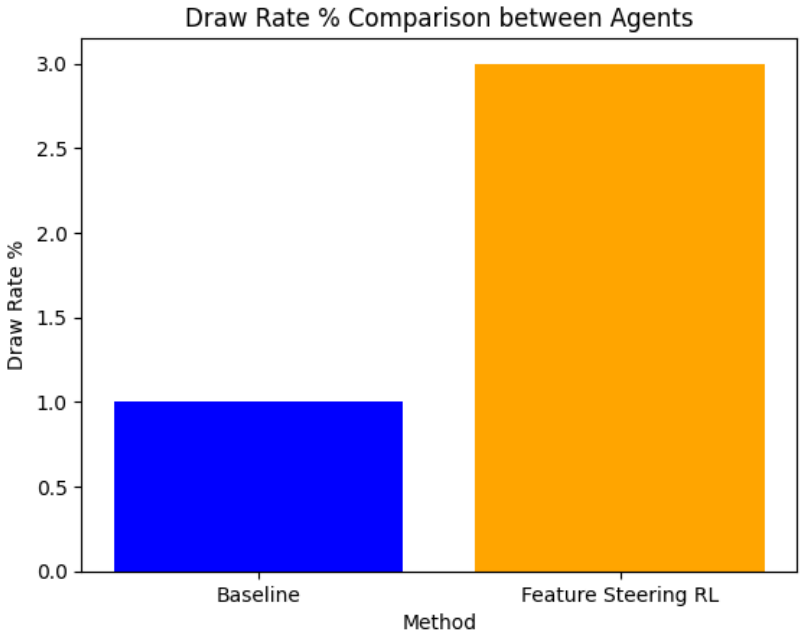


Figure 2: Draw rate of our models against the optimal policy. Our RL agent is able to achieve a 3X improvement over the baseline model.

with graduated penalties and limiting the steering range of features, we achieved faster convergence to desired behaviors. This progression mirrors principles of curriculum learning, where models are trained on increasingly complex tasks to facilitate more efficient learning.

Interestingly, our approach introduces a theoretical trade-off in problem complexity. For simple tasks like Tic Tac Toe, where the original action space is categorical and low-dimensional, feature steering actually increases the complexity by requiring the agent to learn optimal values for multiple continuous features. However, this same characteristic could potentially simplify more complex domains. In robotic control tasks, for instance, where actions typically involve numerous continuous actuator values, a well-calibrated feature space might offer a more compact and manageable representation of the action space.

This feature steering approach stands in stark contrast to traditional fine-tuning methods. While fine-tuning directly modifies model weights through backpropagation, potentially disrupting learned features and compromising interpretability, our method preserves the model's internal structure while achieving targeted behavior modifications through controlled feature manipulation. This preservation of model internals is particularly valuable for maintaining the utility of previously discovered interpretable features.

Several promising directions exist for future research. First, a comprehensive hyperparameter search could substantially improve performance beyond our initial results. Testing with more capable models, such as Llama 3 70B, could reveal how our method scales with model size. Additionally, exploring different environments—from traditional RL benchmarks to RLHF scenarios—would help establish the breadth of our method's applicability. A particularly interesting avenue would be deeper integration of the RL algorithm with the model, potentially using certain activations as additional input alongside environment states. This could lead to a more context-aware steering mechanism capable of automatically adapting feature modifications across diverse tasks beyond classic RL environments.

In conclusion, our work demonstrates the potential of combining RL with Activation Steering for interpretable model reprogramming. While our initial results are modest, the method's transparency, flexibility, and preservation of model internals offer significant theoretical advantages over traditional fine-tuning approaches. As AI systems become increasingly complex and widespread, techniques that enable precise, interpretable behavioral modifications while maintaining model stability will become increasingly valuable. Our approach represents a step toward more controlled and understood methods of AI system modification, potentially contributing to the broader goal of developing more reliable and trustworthy AI systems.

References

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., Chen, C., Olsson, C., Olah, C., Hernandez, D., Drain, D., Ganguli, D., Li, D., Tran-Johnson, E., Perez, E., ... Kaplan, J. (2022). Constitutional ai: Harmlessness from ai feedback. https://arxiv.org/abs/2212.08073

Cunningham, H., Ewart, A., Riggs, L., Huben, R., & Sharkey, L. (2023). Sparse autoencoders find highly interpretable features in language models. https://arxiv.org/abs/2309.08600

Goodfire. (2024). Goodfire api [Accessed: 2024-11-24].

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., & Lowe, R. (2022). Training language models to follow instructions with human feedback. https://arxiv.org/abs/2203.02155

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. https://arxiv.org/abs/1707.06347

Turner, A. M., Thiergart, L., Leech, G., Udell, D., Vazquez, J. J., Mini, U., & MacDiarmid, M. (2024). Steering language models with activation engineering. https://arxiv.org/abs/2308.10248

6 Appendix

6.1 System Prompt

We make use of 5 shot prompting to give examples of how the game is displayed to the model. We use a minimal board representation following the findings mentioned in this blog https://1a3orn.com/sub/essays-tic-tac-toe.html where models demonstrated the highest success rate when the board was simple.

Here is the prompt that we use:

```
You are playing a game of Tic-tac-toe. You must place your symbol in an empty cell to win.
Win by getting 3 of your symbols in a row (horizontal, vertical, or diagonal).
You can only place your symbol in empty cells (numbered 1-9).
Only output the number of your chosen move.

Example valid moves and outcomes:

Example Board 1:
1 2 3
4 5 6
7 8 9
You are X.
X plays 1 -> Output: 1

Example Board 2:
X 2 3
4 5 6
7 8 9
You are O.
O plays 3 -> Output: 3

Example Board 3:
X 2 O
4 5 6
7 8 9
You are X.
X plays 5 -> Output: 5
```

```
Example Board 4:
X 2 O
4 X 6
7 8 O
You are O.
O plays 7 -> Output: 7

Example Board 5:
X 2 O
4 X 6
O 8 9
You are X.
O plays 9 -> Output: 9
```

6.2 User Prompt

The user prompt gives information about the current state to the model. There are two fields in the string that must be substituted programmatically.

Here is the prompt that we use:

```
Now here is the current board:
{board}
You are {player_type}

Output your move (1-9):
```

6.3 Converting Model Output to Move

It is difficult to get an LLM to output a single number directly. More often than not, the model will output irrelevant text along with the number. To extract the number, we use a regular expression to match the number in the output. We can then simply pass this number to the environment to make the move.

6.4 Hyper-Parameters

Determining the hyper-parameters and rewards for the RL agent was a challenging task. We are quite certain that the current handpicked setup is nowhere near optimal which explains our relatively poor performance. Here are the hyper-parameters that we used in Table 1. If a value is not mentioned, it is the default value of the library.

6.5 Action Space

As described in the main text, we extract the most relevant features of the token corresponding to a valid move. Here are the top features in Table 2 that were extracted after 100 games:

6.6 Environment Feature Visualization

We created a user interface to depict the most important features that were used by the RL algorithm during steering. We define importance by the absolute value of the mean activation of a feature in a state. The creation of this interface enables us understand the inner workings of the model and how it is being steered. An example of an interpretable behaviour can be examined in Figure 3. We see there is a strong penalty assigned to the 'Activation on the numeral 5' feature. When examining the board, we see that this spot (centre) is already occupied which makes it an illegal move and returns reduced rewards. Simultaneously, there is a strong push towards the 'The number 7' feature (bottom-left) which is one of the optimal moves in the game for this state. We invite you to test the tool yourself by running the visualize script in our repository.

Hyper-parameter	Value
Number of parallel environments	12
Learning rate	0.00001
Number of games	1
Number of actions (SAE)	30
Number of training steps	6000
Retry count for LLM	2
Steering range for SAE features	[-0.25, 0.25]
Reward for generating non-sensical text	-10
Reward for generating invalid move	-5
Reward magnitude for losing	-2
Reward for getting a draw	10
Reward for playing an optimal move	1
PPO batch size	24
PPO network architecture	3 MLP layers of 100
PPO steps before update	24

Table 1: Hyper-parameters used in the training

Feature
Small positive integers (1-23) in code and data structures
Activation on the numeral 5, especially in scientific contexts
The model's turn to respond in ASCII-based games
May dates in SQL queries
Numerical patterns and sequences in structured data
Representation of the number 2 or second position
Detection of the numeral 2
Grid-based representations in games and visual patterns
Numbered list items in language processing tasks
The number 4
Small to medium numbers in structured lists or game results
ASCII art vertical lines and forward slashes
The model should complete a code snippet for a simple game implementation
Sequences or groups of consecutive numbers
The number 7
Sequential number listing activation
Numbers ending in 8 in sequential lists
The number 8 in technical or quantitative contexts
The number 6 in sequences or lists
Small integer and decimal representation
ASCII art structural elements (vertical lines and box-drawing characters)
Upper bounds in programming examples (often 100 or 200)
Time representation in schedules and daily life
Recognition of the number 9 in scientific contexts
Sales Performance Metrics and Comparisons
Recognition of the number 3 (digit or word)
Detecting win-checking code in game implementations
Recognition of '3' or 'Three' in 3D-related contexts
Describing core gameplay mechanics and scoring in sports

Table 2: Features extracted from 100 games with the baseline model. We observe that several features correspond to numbers. However, there are also other features that are more abstract and are not directly related to numbers.

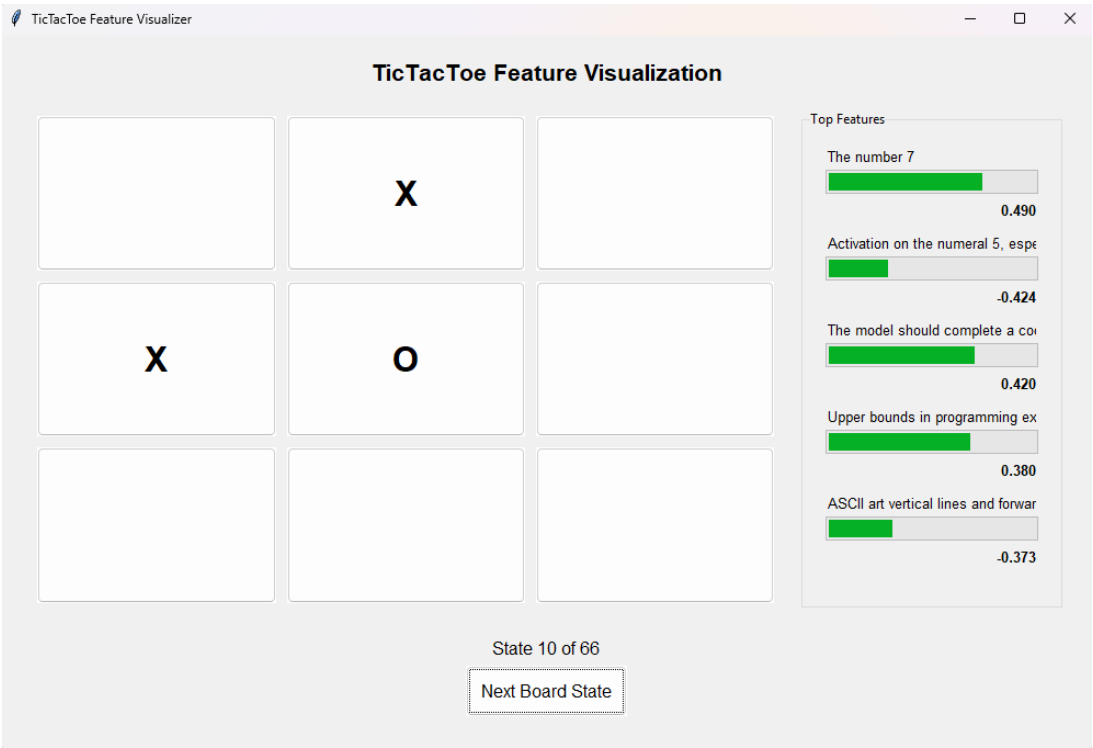


Figure 3: Feature visualization of states.