Fine-Tuning and Merging Large Language Models for Strategic Games

Ian Hughes

Master's Thesis

January 2025

Supervisor 1: Dr Vaios Laschos (WIAS)
Supervisor 2: Prof Martin Skutella (TUB)

Statutory Declaration of Authorship:

I hereby declare that the thesis submitted is my own, unaided work, completed without any external help. Only the sources and resources listed were used. All passages taken from the sources and aids used, either unchanged or paraphrased, have been marked as such. Where generative AI tools were used, I have indicated the product name, manufacturer, the software version used, as well as the respective purpose (e.g. checking and improving language in the texts, systematic research). I am fully responsible for the selection, adoption, and all results of the AI-generated output I use. I have taken note of the Principles for Ensuring Good Research Practice at TU Berlin dated 8 March 2017: `https://www.tu.berlin/en/working-at-tu-berlin/important-documents/guidelinesdirectives/principles-forensuring-good-research-practice`

I further declare that I have not submitted the thesis in the same or similar form to any other examination authority.

[illegible] 16.1.2025

Ian Hughes Date

Abstract

With the recent major research and practical usage of Large Language Models (LLMs), one question that remains is how effectively they can reason and be trained in reasoning tasks. The goal of this work is to explore the ability of small, open-source LLMs to play strategic games. Further, we explore fine-tuning these LLMs in strategic game-play using Supervised Fine-Tuning and Direct Preference Optimization. A subset of the fine-tuned models are also compared to model created with three model merging techniques: Linear [1], Spherical Linear Interpolation [2], and Model Stock [3] merging. To enable this exploration, we implement improvements to a benchmarking suite for LLMs, named GT-Bench, that measures LLM performance across ten strategic games [4]. The results from our small LLM game-play experiments line-up with the results from the original GT-Bench paper. Further, our research shows that fine-tuning with Direct Preference Optimization can be successful for games such as *Tic-Tac-Toe* and *Prisoner's Dilemma*, but may fall short for complex games or strategies. The model merging methods provided mixed results, with success in *Prisoner's Dilemma* but no game-play improvement in *Tic-Tac-Toe*.

Contents

Chapter 1

Introduction

Large language models (LLMs) have exceptional abilities in natural language modeling and have recently gone beyond their initial scope with proficiency in programming, scientific knowledge, and finance [5]. One question that remains is how LLMs perform in logic and reason based tasks, and further, how well they can be adapted or fine-tuned to such tasks. While research has quickly expanded in the field of LLMs, with nearly 30,000 papers referencing LLMs in 2024, there is still work remaining to understand the emergent capabilities of LLMs in strategic reasoning [5]. In this work, we look to analyze the performance of LLMs in 10 strategic games against a variety of opponents including other LLMs, a random decision agent, and a Monte-Carlo-Tree-Search algorithm agent. Beyond this, we will compare and contrast the performance of the base model LLMs to LLMs that have been fine-tuned to play strategic games. We will use both the supervised fine-tune and Direct Preference Optimization methods, the latter of which uses reinforcement learning from human feedback, to fine-tune the LLMs. Model merging of various fine-tuned models will also be performed in order to analyze how model merging differs from fine-tuning with respect to strategic game-play performance. Beyond strategic performance, we will fine-tune the LLMs to specific strategies and then measure their adherence rate to these strategies. Performing these model trainings and evaluations in the context of strategic games and decision making is novel given that most prior works focus on baseline game-play with pre-trained (base) LLMs or on improving strategic play with specific prompting styles. This work uses a strategic game benchmark, GT-Bench, to evaluate various LLMs consistently across the 10 games [4]. As will be detailed in this paper, GT-Bench provides the foundation for this work. LLM fine-tuning, model merging, custom LLM API integration, and additional commercial (Google) LLMs were added ad-hoc or directly into the GT-Bench codebase in order to conduct our experiments. The ultimate goal of this work is to compare different fine-tuning and model merging methods within the GT-Bench framework of strategic games and reasoning.

Remark 1.0.1. From now on we may refer to a single Large Language Model as a ‘LLM’ and multiple Large Language Models as a ’LLMs’.

Chapter 2

Definitions

In this chapter we provide a subset of LLM definitions and acronyms to aid in the understanding of the terms used throughout this paper.

Large Language Models (LLMs) are artificial intelligence systems which can process and generate text for a variety of tasks such as chatting, code generation, game-play, and more [5].
A **Token** is a LLM's representation of text. Each LLM will have a collection of numeric tokens which represent words or sub-words [5].
Temperature is the LLM parameter responsible for randomness in next-token generation for the LLMs, with values typically falling in the range: $[0, 2]$ [6]. A value closer to 0 would lead to a more deterministic next-token generation.
Fine-Tuning is a LLM training process used to align an LLM to a specific task such as coding or domain specific text generation. This process occurs after pre-training, which is the process used to create the initial or base LLM from web-scale level training data [5].
The **Learning Rate** is utilized in the fine-tuning process. This hyper-parameter controls the rate at which the LLM adapts to the training dataset and a small learning rate is generally needed to have training stability [5].
An **Epoch** is a complete cycle, or full backward and forward pass, of training through the fine-tuning dataset [7].
Proximal Policy Optimization (PPO) is the alignment method for Reinforcement Learning from Human Feedback (RLHF), which uses a reward model to rank different LLM-generated responses [5].
Direct Policy Optimization (DPO) is an alternative RLHF alignment

method which uses an implicit reward model to rank LLM-generated responses [8].
Supervised Fine-Tuning (SFT) is the term for a fine-tuning or LLM alignment which requires only context (or input) and response (or output) pairs for training. SFT is less computationally intensive than RLHF methods (PPO and DPO), and can be just as effective in fine-tuning under certain conditions [5].
Model Merging combines, or merges, the task/weight vectors from two or more LLMs with the goal of creating improved or diversified responses in the new, merged LLM [9].
Zero-Shot Prompting is a LLM prompting style which does not use additional examples or context - only the prompt/question is given to the LLM [5].
Chain-of-Thought Prompting encourages LLMs to list their responses to problems or inquiries in a step-by-step manner. This can be done in an effort to increase their accuracy in reasoning or complex tasks [10].

Chapter 3

Large Language Model and GT-Bench Background

In this chapter, we will give an introduction to Large Language Models and then discuss the major results given in the GT-Bench paper. Next, an introduction to the specific LLMs used in this work will be presented. Lastly, the motivation and goals of this work will be listed.

3.1 Large Language Model Review

3.1.1 Introduction

Statistical language modeling dates back to the 1950s with n-gram language models and has since become fundamental to a number of natural language modeling tasks such as speech recognition, translation, and information retrieval [11]. A large improvement in language modeling occurred in 2014 with the sequence to sequence learning model which used Recurrent Neural Networks (RNNs) as well as a multi-layered Long Short-Term Memory input sequence to vector mapping to generate English to French translations [12]. This method is close to the paradigm of current LLMs where there is a pre-training and fine-tuning stage. When using this process, the final model can be used for a wider variety of tasks when compared to early language models [11]. The transformer architecture marked another large milestone in modern LLM progress. With this architecture, self-attention is applied in parallel for each word in a training sample to create an ”attention score”, which models the impact that words have on each other [13]. Transformers allow for much greater parallelization than RNNs which has greatly increased the efficiency of pre-training LLMs on large amounts of data with Graphic Processing Units (GPUs) [11]. As access to LLMs has increased, researchers, and even hobbyists, now utilize supervised fine-tuning (SFT) to align LLMs with a specific task by using labeled training data [7]. This SFT process can be potentially costly depending on the dataset size. SFT is also a step used prior to reinforcement leaning in LLMs. A notable advancement in reinforcement learning was given by the ChatGPT team, who used Reinforcement Learning from Human Feedback (RLHF) to vastly improve the LLM's ability to follow human instructions [14]. Pre-training on web-scale text combined with RLHF has allowed LLMs to follow instructions for tasks of increasing complexity [11]. In this work, we will utilize both SFT and RLHF fine-tuning methods to increase the performance of fine-tuned LLMs in game-based reasoning.

3.1.2 Large Language Model Reasoning Capabilities

While LLMs have made significant improvements in reasoning based tasks, it is not clear whether LLMs are reasoning or not [15]. For example, when looking at Chain-of-Thought prompting, we see the potential for reasoning and step-by-step thinking in LLM responses, but it is unclear whether the neural network is reasoning or simply emulating the reasoning process [10]. To further explore the question of reasoning in LLMs, many researchers have turned to games to examine the strategic capabilities of these models in a more structured format. It was shown that human-level play was possible in the game of *Diplomacy* with the combination of a language model and a reinforcement learning mechanism. This hybrid model was in the top 10% of players in *Diplomacy* [16]. Using game theoretic evaluations in the games of *Prisoner's Dilemma* and *Battle of the Sexes*, we see that LLMs can succeed when playing games that require a selfish strategy but struggle in games of cooperation [17]. For these games, performance was also improved when prompting for the other player's suspected strategy or for cooperation in a game. When playing text games, in which a player must respond to situations and understand dialogue, ChatGPT 4 can play competitively but has a very low intelligence and much lower skill level than human players [18]. Specifically, encountering unknown scenarios leads to repetition in lieu of predicting a potentially incorrect answer. Experiments involving several

strategic negotiation games (like *Deal or no Deal*) and a systematic Chain-of-Thought prompting style gave LLMs near human negotiation skills [19]. In this instance, the Chain-of-Thought prompting revealed payoffs and potential strategies to the LLM to aid in decision making. One limitation mentioned in this work, though, was the drop in performance once the complexity of the games went beyond the context window of the utilized LLMs. We find a similar issue with fine-tuning LLMs in this work, where an increase in game complexity leads to fine-tuning difficulties.
Other works exploring the theme of LLMs and strategic game-play include ChessGPT, which was an effort to standardize LLM text inputs from Chess matches and to fine-tune an LLM on different chess positions and strategies [20]. While the fine-tuning in ChessGPT improved the LLM's understanding of different board positions, there was very little improvement in game-play for various endgame and strategic scenarios. Given the complexity of chess, and the relatively small LLM being trained in ChessGPT, the results lineup with the conclusions that we will present later in this paper surrounding the inability of relatively small, fine-tuned LLMs to succeed in complex games. This was one of the few academic papers we found which focused on fine-tuning an LLM in strategic game-play. While not an LLM model, work by Google Deep Mind showed the ability of a model with the transformer architecture, shared by LLMs, to play chess at the Grandmaster level without using a search algorithm [21]. While the pre-training of the Deep Mind chess model and the LLMs used in this work differ, this proved the ability of the transformer architecture to succeed in a complex game with a vast decision space.

3.2 GT-Bench Result Summary

GT-Bench is a benchmark for LLMs in ten unique games and the accompanying paper provides insights into the baseline abilities of various LLMs in strategic games [4]. Four of the games in GT-Bench are complete information and deterministic games and six are incomplete information and/or random outcome games. The overall goal of the GT-Bench research was to create a standard benchmarking tool for the game theoretic evaluation of LLMs. The GT-Bench work also included evaluations of LLMs including but not limited to: Chat GPT 3.5 and 4, Llama 2 34b and 70b, and Mistral 7B. The benchmark allows for a wider examination, when compared to prior

works, of strategic LLM performance. Also, the standardized prompt formatting and use of a standard game engine allows for further development and contributions to the project.
We breakdown the ten games in GT-Bench between complete information and deterministic games (4) versus probabilistic and/or incomplete information games (6):

- Complete and Deterministic: *Tic-Tac-Toe*, *Connect-4*, *Breakthrough* and *Nim*
- Probabilistic and/or Incomplete Information: *Kuhn Poker*, *Liar's Dice*, *Negotiation*, *Pig*, *Prisoner's Dilemma* and *Blind Auction*

Further descriptions of *Tic-Tac-Toe*, *Prisoner's Dilemma* and *Kuhn Poker*, which are the subjects of the fine-tuning and model merging experiments in this work, can be found in Section 6.1. For all other games, full descriptions and prompt examples can be found in the GT-Bench paper [4].
The potential opponents for each LLM include a Random agent, a Monte-Carlo-Tree-Search (MCTS) agent, a tit-for-tat agent (*Prisoner's Dilemma* only), and other LLM agents. The Random agent selects randomly from the set of all legal moves. A simple measure of a LLM's performance is whether or not it outperforms the Random agent. The tit-for-tat strategy is to mimic your opponent's prior round move [4]. For example, in *Prisoner's Dilemma* if player 2 testified in round 3 then the tit-for-tat strategy for player 1 would be to testify in round 4. The MCTS agent uses the Monte-Carlo-Tree-Search algorithm, which is a decision making algorithm that recursively simulates game states across the branches and nodes of a game based decision tree [22]. This algorithm was state-of-the-art for playing the game *Go* upon its inception in 2006, and remains highly performative and popular in many complete and deterministic games. We would not expect the LLMs to compete against the MCTS agent in complete and deterministic games given the capabilities and design of the MCTS algorithm.
The major findings from the GT-Bench paper are as follows:

1. LLM agents perform better in incomplete information and/or random outcome games when compared to complete information and deterministic games.
2. LLM agents are not competitive against MCTS agents in complete information and deterministic games.

3. When compared to zero-shot prompting (base prompting), sophisticated prompting styles such-as Chain-of-Thought reasoning do not always increase the performance of LLMs.

4. Code/Instruction pre-training of LLMs tends to increase game performance and reasoning when compared to conversational or non code pre-trained LLMs.

5. Open-source LLMs generally perform worse than commercial LLMs in the games.

Given these outcomes, our work looks to focus on instruction pre-trained LLMs as the base models for fine-tuning. Further, given the finding that Chain-of-Thought prompting or other reasoning methods are not always advantageous, we will only use the base (zero-shot) prompting in our evaluations. While this will not allow us to explore the impact of fine-tuning on different prompting styles, computational limitations make it necessary to limit the number of experiment permutations in this work. Given the finding that open-source LLMs are generally worse than commercial LLMs, we will focus on the open-source LLMs and the potential fine-tuning or model merging techniques that could make them outperform commercial LLMs. Beyond this, more control is available for fine-tuning open-source LLMs when compared to commercial options, which are both expensive and rigid in their training options. Further work could explore commercial fine-tuning options and compare them with the open-source options utilized in this paper.

3.3 Models Utilized

Three major LLM families will be used in this work: Llama from Meta, Phi from Microsoft, and Gemini from Google. Llama is a family of LLM models exclusively trained on publicly available datasets, with the larger parameter models (405 billion parameters) being competitive with commercial models such as ChatGPT 4 and Gemini [23]. For this work, we used the Llama 3.1 8b Instruct model, which is one of the smallest LLMs in the Llama family. This 8 billion parameter Llama model has been further tuned for coding and human instruction by the Llama team. Phi is an open-source model family from Microsoft touted for its high benchmark scores with respect to its relatively small parameter size [24]. In this work, we utilize Phi 3.5 Mini Instruct, which has 3.8 billion parameters and has been further tuned for coding and human instruction. Gemini represents the state-of-the-art commercial model from Google. Gemini is currently implemented in Google search and has benchmarks similar to or exceeding ChatGPT 4 in many categories [25]. Phi 3.5 Mini Instruct and Llama 3.1 8b Instruct were both fine-tuned in this work while the Gemini models were used as further benchmark comparisons and LLM opponents. A list of the base and fine-tuned models used in this work can be found in Section 7.5.

3.4 Motivation and Research Goals

The goal of this work is to compare fine-tuning and model merging methods for LLM performance and strategy adherence in strategic games. While our review of literature showed many researchers using base LLMs to evaluate the efficacy of LLMs' strategic reasoning ability, there were very few instances of fine-tuning LLMs with strategic game-play prompt and response pairs. Questions that we are looking to answer include the following:

1. How does fine-tuning impact the performance of small LLMs in strategic games?
2. Does fine-tuning in one game improve performance in other games?
3. How do different datasets impact fine-tuning?
4. How does RLHF with DPO compare to SFT with respect to benchmark performance?
5. Are there conditions where merging two fine-tuned models is a suitable alternative to an additional fine-tuning over a larger dataset?

To achieve our goal and answer these questions, we modified the GT-Bench codebase to generate training data and allow custom LLMs. Fine-tuning was then done for several games and dataset formations to investigate what datasets and methods would lead to performance and/or strategy adherence improvements. After creating three merged models, we compared a subset of the fine-tuned LLMs to the merged LLMs.

Chapter 4

Large Language Model Fine-Tuning and Merging Methods

In this chapter we will introduce the integral components of the fine-tuning and model merging methods used in this work. Our focus will begin with an overview of the Low-Rank Adaptation method for fine-tuning LLMs. This will be followed by an overview of Direct Preference Optimization (DPO), which is a RLHF method for fine-tuning LLMs. Finally, we will give a brief overview of model merging and review the three merging methods employed in this work.

4.1 Low-Rank Adaptation of Large Language Models

The Low-Rank Adaptation (LoRA) of LLMs allows for an efficient fine-tune of a LLM while modifying only a subset of the model parameters. The notation, methodological summaries, and equations in this section come from the original LoRA paper [26]. LoRA is reviewed here given that the SFT and DPO processes both use LoRA to reduce the processing time and memory overhead necessary to fine-tune our LLMs. All of the models we produced also benefited from the storage reduction that fine-tuning with LoRA brings. While the LoRA process could be generalized to other models as well, the authors' motivating use case for LoRA was with language modeling.

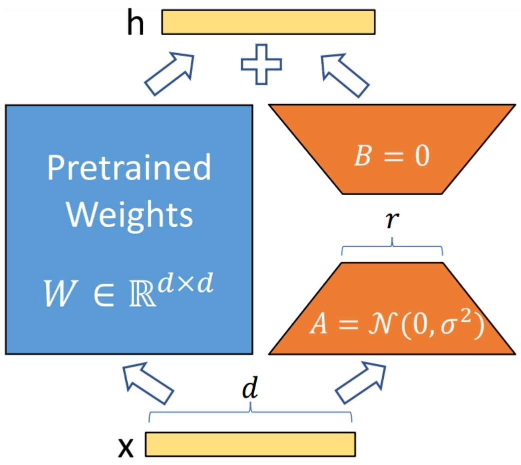


Figure 4.1: Visualization from [26] showing the low-rank update with matrices A and B versus a full-rank update with W. In [26] the authors note the following: "Using GPT-3 175B as an example, we show that a very low rank (i.e., r in [this figure] can be one or two) suffices even when the full rank (i.e., d) is as high as 12,288, making LoRA both storage- and compute-efficient."

LoRA freezes pre-trained LLM weights while training a subset of dense layers in the LLM with optimization over low-rank decomposition matrices [26]. This low-rank decomposition can be visualized in Figure 4.1 where the full-rank update matrix, W, will be significantly larger then our low-rank encoding using matrices A and B.
Now that we have visualized the LoRA method, we will go back to the original LoRA problem statement. We begin with a language model $P_\Phi(y|x)$, where Φ parametrizes our language model. $P_\Phi(y|x)$ could be any LLM which uses the transformer architecture. Our goal may be to adapt this language model to specific tasks like coding, conversation, or in our case, strategic game-play. Such adaptation requires a training dataset of input and target pairs: $\mathcal{Z} = \{(x_i, y_i)\}_{i=1,..,N}$. In this formulation, x_i and y_i are LLM token sequences representing the input (or context) and target, respectively. For our strategic game-play fine-tuning task, x_i would represent a game-play prompt while y_i would give the associated game move. This same context and target paradigm can be applied to other LLM tasks like code or chat generation.
For a complete or full fine-tune, the model would be initialized using the base model weights, Φ_0. Next, Φ_0 would be updated to $\Phi_0 + \Delta\Phi$ by "repeatedly

applying the gradient to maximize the language modeling objective" [26]:

$$\max_{\Phi} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(P_{\Phi}(y_t|x, y_{<t})\right) \tag{4.1}$$

In a full fine-tune, we would acquire a set of unique parameters, $\Delta\Phi$, for each text generation task. A potential issue with this approach is that the dimension for $\Delta\Phi$ is equal to Φ_0 [26]. This computational overhead is a major drawback to performing a full fine-tune. Many LLM models have billions (or even trillions) of parameters, so the storage and deployment of these instances could be challenging and potentially infeasible. LoRA encodes the fine-tune specific parameter increment $\Delta\Phi = \Delta\Phi(\Theta)$ by Θ, where $|\Theta| \ll |\Phi_0|$ [26]. This allows us to find $\Delta\Phi$ by optimizing over Θ [26]:

$$\max_{\Theta} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(p_{\Phi_0+\Delta\Phi(\Theta)}(y_t|x, y_{<t})\right) \tag{4.2}$$

Figure 4.1 illustrates the low-rank representation of $\Delta\Phi$ while the full details for the decomposition and encoding can be found in [26].
LoRA is utilized in all of our fine-tunes of Phi 3.5 and Llama 8.1. Using LoRA allowed us to fine-tune these relatively small LLMs on a single GPU, which made processing cost effective and efficient for all of our model permutations.

4.2 Direct Preference Optimization

In this section we go over the relevant background of Direct Preference Optimization (DPO), a RLHF method to fine-tune LLMs. The notation, methodological summaries, and equations in this section come from the original DPO paper [8]. RLHF uses preference data and a reward model to train an LLM with reinforcement learning. In contrast to the original RLHF method using Proximal Policy Optimization (PPO), which uses an explicit reward, DPO utilizes an implicit reward model. This allows the model to operate without costly human labeling or the creation of an additional reward model. A visual summary of how DPO uses an implicit versus explicit reward model can be seen in Figure 4.2. Whether using DPO or PPO, it is advised to first perform a SFT on the LLM in order to create a reference policy which is aligned with the final reward model.

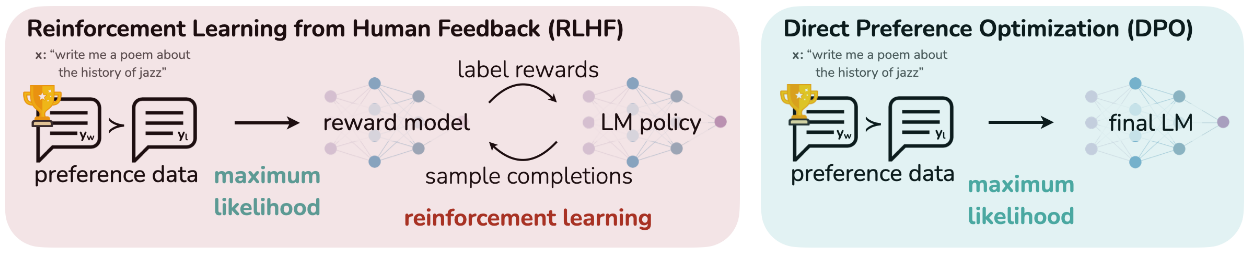

Reinforcement Learning from Human Feedback (RLHF)
x: "write me a poem about the history of jazz"
preference data
maximum likelihood
reward model
label rewards
LM policy
sample completions
reinforcement learning
Direct Preference Optimization (DPO)
x: "write me a poem about the history of jazz"
preference data
maximum likelihood
final LM

Figure 4.2: Rather than using reinforcement learning, DPO relies on an implicit reward model. This allows to DPO to optimize for human preferences without having a separate reward model and costly human reward labeling phase. Only a preferred and rejected response are needed in DPO, with no explicit reward assigned. This figure comes from [8].

We begin our discussion of the DPO reward model within the reinforcement learning framework. In this framework, we optimize over an objective function using policies from our target model π_θ and reference model π_{ref}, as well as a reward model, $r_\phi(x, y)$, which is based on preference data. Within the reinforcement learning framework, we use a reward function to provide feedback to our language model. Inline with prior works, the RLHF optimization equation, with prompts x and responses y, is as follows:

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(y|x)}\big[r_\phi(x, y)\big] - \beta\mathbb{D}_{\text{KL}}\big[\pi_\theta(y \mid x) \mid\mid \pi_{\text{ref}}(y \mid x)\big], \tag{4.3}$$

where the parameter β controls how much deviation we have from the reference policy π_{ref} [8]. The base reference policy, π_{ref}, is advised to come directly from our SFT model. This function is discrete and non-differentiable so the usual approach is to optimize with reinforcement learning. Prior to DPO, the standard approach was to take the reward function,
$r(x, y) = r_\phi(x, y) - \beta(\log \pi_\theta(y \mid x) - \log \pi_{\text{ref}}(y \mid x))$, and maximize using PPO [27]. DPO begins with this same reinforcement learning objective, but uses reparameterization to remove the explicit reward function from the preference model.
The authors of the paper demonstrate that the Bradley-Terry preference equation, which is used in the reward model, can be rewritten entirely in terms of the preference data and optimal policy using our target model, π_θ, and our reference model, π_{ref}. The full details of this derivation are given in [8]. Rewriting the optimization in terms of an implicit reward model results

in the following equation, with y_w and y_l representing the chosen and rejected responses, respectively [8]:

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta\log\frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right)\right]. \tag{4.4}$$

Taking the gradient of the above equation with respect to θ allows us to visualize the DPO update step piece-wise,

$$\begin{aligned}&\nabla_\theta\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = \\ &-\beta\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\Bigg[\underbrace{\sigma(\hat{r}_\theta(x,y_l) - \hat{r}_\theta(x,y_w))}_{\text{higher weight when reward estimate is wrong}}\Bigg[\underbrace{\nabla_\theta\log\pi(y_w \mid x)}_{\text{increase likelihood of } y_w} - \underbrace{\nabla_\theta\log\pi(y_l \mid x)}_{\text{decrease likelihood of } y_l}\Bigg]\Bigg],\end{aligned}$$

where our reward $\hat{r}_\theta(x,y) = \beta\log\frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$ is implicitly defined by the language model we are fine-tuning, π_θ, and our reference model, π_{ref} [8].
In our work, fine-tuning with DPO was highly dependent on the datasets used in the training process. That is, once the hyper-parameters of our training resulted in a stable training, the choice and frequency of the chosen and rejected responses, y_w and y_l, had a large impact on the output and performance of our fine-tuned LLMs.

4.3 Model Merging

Model merging combines, or merges, the task or weight vectors from two or more different models using addition, negation, and/or task vector analogies with the goal of creating an improved or diversified response in the new merged model [9]. We will focus on model merging from the same base model using addition. Our model merging goal is to assess the differences between fine-tuning on a larger dataset for multiple games versus merging the output from two fine-tuned LLMs. Beyond this, we would like to analyze the effectiveness of model merging on complex versus more basic game strategies.

4.3.1 Linear Model Merge

Linear model merging adds the model parameters of two or more LLMs using a weighted average. When introduced, this method suggested a simple average of model parameters from the top N models which were fine-tuned on a specific image classification task, while validation was also done in text classification models [1]. The MergeKit Python package implementation, which was used in this work, is more general and allows for a weighted average of model parameters (or weights) between N LLMs [28]. An adapted version of the MergeKit implementation is given in Algorithm 1.

Algorithm 1 Linear Model Merge

```
1: in $p$: Vectors of model parameters (or weights) for $N$ LLMs being merged, $w$: Weighting factor for $N$ LLMs being merged, $Norm$: Boolean to normalize the final merged weight vectors (normalization recommended)
2: $p_{merge} = \sum_{i=1}^{N} p_i \, w_i$
3: if $Norm$ = True then
4:     $p_{merge} = p_{merge} \, / \, \sum_{i=1}^{N} w_i$
5: end if
6: out $p_{merge}$
```

4.3.2 SLERP Merge

Spherical Linear Interpolation (SLERP) was originally developed for use in computer graphics and animation in order to smoothly interpolate sequences of arbitrary rotations using curves over a sphere [2]. When applied to LLM merging, SLERP is used to spherically interpolate the parameters of two different models using the angle between them [28]. An adapted of the MergeKit implementation is given in Algorithm 2.

4.3.3 Model Stock Merge

The motivation for the Model Stock merge method comes from two fundamental findings from the method's authors [3]:

1. Fine-tuned model weights lie on a thin shell in the weight space, layer-wise.

2. Fine-tuned weights with a closer proximity to the center of the thin shell, μ, give a better ImageNet Benchmark classification accuracy.

Algorithm 2 SLERP Model Merge

```
1: in $v_1$: Model parameter vector for 1st of 2 LLMs being merged, $v_2$: Model parameter vector for 2nd of 2 LLMs being merged, $t$: Float value between 0.0 and 1.0 to control how far on the arc between $v_1$ and $v_2$ we traverse.
2: $v_1 = norm(v_1)$
3: $v_2 = norm(v_2)$
4: $v_{dot} = v_1 \cdot v_2$
5: $\theta = \arccos(v_{dot})$ {Calculate initial angle between $v_1$ and $v_2$}
6: $\theta_{\sin} = \sin(\theta)$
7: $\theta_t = \theta * t$ {Angle at time-step $t$}
8: $\theta_{t\sin} = \sin(\theta_t)$
9: $s_1 = \sin(\theta - \theta_t)/\theta_{\sin}$ {Apply the SLERP algorithm}
10: $s_2 = \theta_{t\sin}/\theta_{\sin}$
11: $res = s_1 * v_1 + s2 * v_2$
12: out $res$
```

Figure 4.3 illustrates how the final merged model weight w_H is found using the weights of two fine-tuned models (w_1 and w_2), the pre-trained model weights (w_0), and the center of the thin shell, μ. This method can also be extended to $N > 2$ fine-tuned LLMs as long as they were fine-tuned from the same base model. Though μ is not initially known, it can be estimated given the geometric relation of w_0, w_1, and w_2 to μ. Proofs and details of these geometric relations are given in [3].

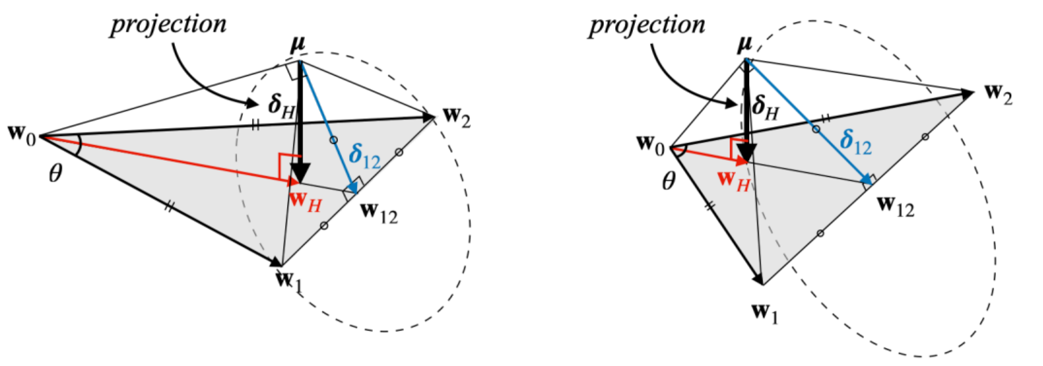


Figure 4.3: This figure and the original caption, which has been rephrased here, comes from [3]. Two scenarios are given in the figure, one with a small angle between the fine-tuned model weights (left) and the other with a large angle between the fine-tuned model weights (right). The span of the pre-trained model weights (w_0) and the fine-tuned model weights (w_1 and w_2) is given by the gray triangle. This gray triangle represents the search space of the three weights with the goal being to find the point on the triangle nearest to the center, μ. The perpendicular foot from μ to the gray triangle gives the nearest point to μ lying in the span of the fine-tuned and pre-trained model weights, marked by the end of w_H.

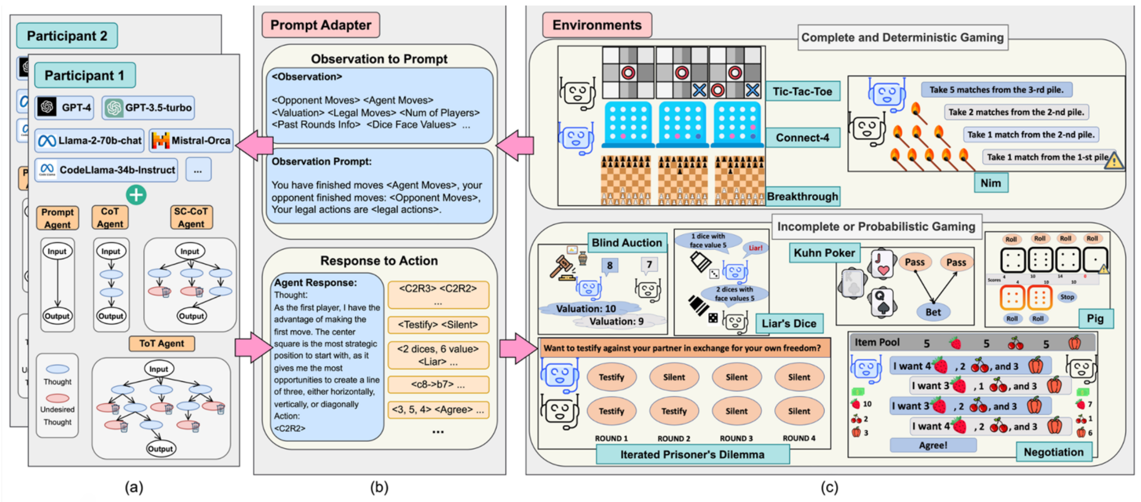
Chapter 5

Code Architecture and Updates to GT-Bench

In this chapter, we give an overview of the original GT-Bench benchmark codebase. After this summary is given, we describe the updates that were made to the codebase in order to support training dataset generation, fine-tuning, model merging, and custom LLM deployment via Google Cloud.

5.1 GT-Bench Codebase Summary

The GT-Bench benchmarking program is the orchestration platform between the LLM prompts, LLM settings, game environments, and LLM API queries. The game environments are created and managed by OpenSpiel, a game engine built by Google Deepmind to support general reinforcement learning and planning in games [29]. OpenSpiel supports each of the ten game environments for GT-Bench, providing the game rules and necessary answer formats for each of the games. The LLM prompts are dynamically created in GT-Bench based on the relevant players and turns of the game. These prompts are then sent to a LLM API endpoint for inference to generate an answer, or move, in the game. GT-Bench then parses the answer given from the LLM allowing the next turn to begin. An illustration of the GT-Bench codebase and processes is given in Figure 5.1.


Participant 2
Participant 1
GPT-4
GPT-3.5-turbo
Llama-2-70b-chat
Mistral-Orca
CodeLlama-34b-Instruct
Prompt Agent
CoT Agent
SC-CoT Agent
ToT Agent
Input
Output
Thought
Undesired Thought
Prompt Adapter
Observation to Prompt
<Observation>
<Opponent Moves> <Agent Moves> <Valuation> <Legal Moves> <Num of Players> <Past Rounds Info> <Dice Face Values> ...
Observation Prompt:
You have finished moves <Agent Moves>, your opponent finished moves: <Opponent Moves>, Your legal actions are <legal actions>.
Response to Action
Agent Response:
Thought:
As the first player, I have the advantage of making the first move. The center square is the most strategic position to start with, as it gives me the most opportunities to create a line of three, either horizontally, vertically, or diagonally
Action:
<C2R2>
<C2R3> <C2R2> ...
<Testify> <Silent>
<2 dices, 6 value> <Liar> ...
<c8->b7> ...
<3, 5, 4> <Agree> ...
Environments
Complete and Deterministic Gaming
Tic-Tac-Toe
Connect-4
Breakthrough
Take 5 matches from the 3-rd pile.
Take 2 matches from the 2-nd pile.
Take 1 match from the 2-nd pile.
Take 1 match from the 1-st pile.
Nim
Incomplete or Probabilistic Gaming
Blind Auction
Valuation: 10
Valuation: 9
Liar's Dice
Kuhn Poker
Pass
Bet
Pig
Want to testify against your partner in exchange for your own freedom?
Testify
Silent
ROUND 1
ROUND 2
ROUND 3
ROUND 4
Iterated Prisoner's Dilemma
Item Pool
I want 4, 2, and 3
I want 3, 1, and 3
I want 3, 2, and 3
I want 4, 2, and 3
Agree!
Negotiation
(a)
(b)
(c)

The overall architecture of GTBENCH. There are three main components from *right* to *left*: **Environments** (c) for game hosting, observation providing, and action execution; **Prompt Adapter** (b) for converting observation to prompt and extracting actions from participants' generations; **Participants** (a) for reasoning and action generation. (SC-)CoT agent refers to (Self-Consistent) Chain-of-Thought Agent and ToT agent refers to Tree-of-Thought agent.

Figure 5.1: Illustration from the original GT-Bench paper giving an overview of the functionality in the codebase [4]

5.2 Code Updates

Several updates were made to GT-Bench in order to support fine-tuned and merged LLMs. First, training data was generated from game prompts using the MCTS (all games except *Prisoner's Dilemma*) or tit-for-tat (*Prisoner's Dilemma*) agent responses. These prompt and response pairs were then formatted and stored in a database. Next, an API connection was made between Google Cloud, where the fine-tuned and merged LLMs were deployed, and GT-Bench. Along with the API update, custom chat code was added to GT-Bench to support the fine-tuned LLMs. Parameters then needed to be added for each new LLM in the GT-Bench program. Finally, a code-block for the Gemini LLMs was added to GT-Bench to support their custom parameters and API calls from Google Cloud. These changes, along with the ad-hoc fine-tuning and model merging processes, are highlighted in Figure 5.2.

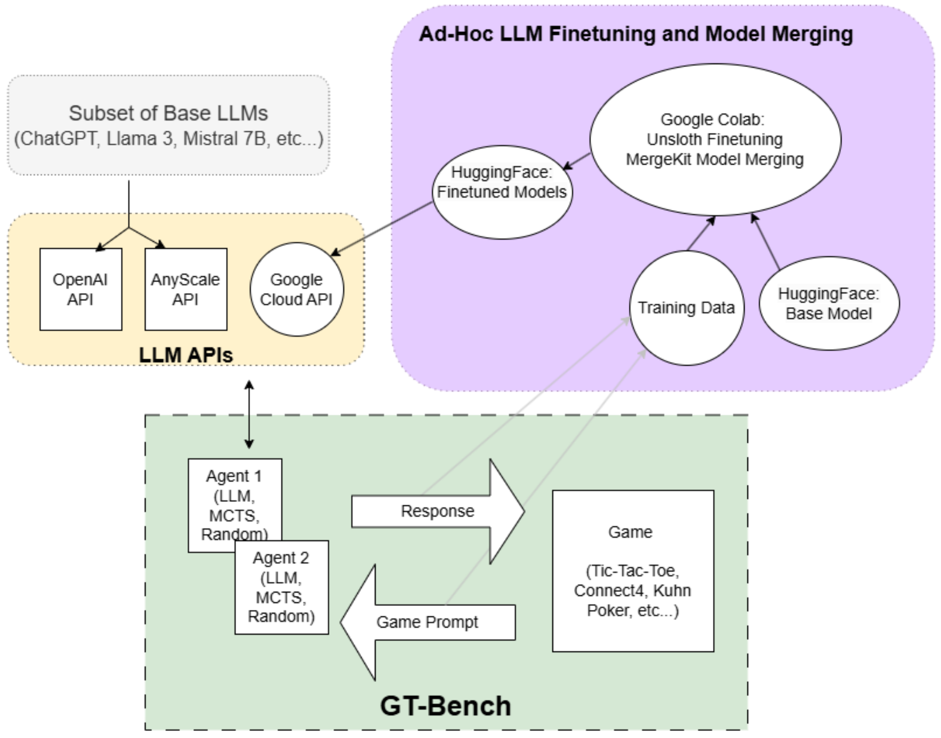

Ad-Hoc LLM Finetuning and Model Merging
Subset of Base LLMs (ChatGPT, Llama 3, Mistral 7B, etc...)
Google Colab: Unsloth Finetuning MergeKit Model Merging
HuggingFace: Finetuned Models
OpenAI API
AnyScale API
Google Cloud API
LLM APIs
Training Data
HuggingFace: Base Model
Agent 1 (LLM, MCTS, Random)
Agent 2 (LLM, MCTS, Random)
Response
Game Prompt
Game (Tic-Tac-Toe, Connect4, Kuhn Poker, etc...)
GT-Bench

Figure 5.2: In this figure we illustrate the major updates made to the GT-Bench codebase. Round shapes indicate new additions, while the square objects represent features existing in the original GT-Bench program.

5.2.1 Training Data Generation

Training data for all of the games, with the exception of *Prisoner's Dilemma*, was generated using the MCTS agent responses to hundreds of games against the Random agent or another MCTS agent. These responses, along with the observation prompts for each move, were then stored in a database. Next, training tables were created by querying the database to create the relevant prompt and MCTS answer pairs. For *Prisoner's Dilemma*, the process was the same as with the MCTS agent except the tit-for-tat agent was used in place of the MCTS agent. The details of the data used in the final training tables for each of the fine-tuning processes is given in Chapter 7.

Counter-Factual Regret Minimization

For *Kuhn Poker*, an additional method was used to generate training data for the DPO fine-tuning process. Counter-Factual Regret Minimization (CFR) starts with a baseline strategy and then iteratively evaluates moves to minimize regret (negative payoffs) using self-play [30]. The goal of this method is to efficiently generate a stable Nash Equilibrium strategy after self-play with $K > 0$ matches. This algorithm was employed for *Kuhn Poker* using 10,000 simulated matches. These simulations were run in a Python notebook created by the OpenSpiel team, which was modified for this work [31]. The output from this notebook, in terms of 'Pass' and 'Bet' move ratios for each prompt, was used for fine-tuning. The goal of this fine-tuning was to test the mixed strategy adherence capabilities of the fine-tuned LLM.

5.2.2 Ad-Hoc LLM Fine-Tuning and Model Merging

Models were fine-tuned using the Unsloth software package's implementation of LoRA, SFT, and DPO [32]. Unsloth provided the templates for fine-tuning in Google Colaboratory (Colab), a lightweight cloud-based Python notebook development platform [33]. All of the fine-tuning was done using updated Unsloth Colab notebooks. The trainings were all performed on either a single Nvidia A100 or L4 GPU processor. Model merging was performed using the MergeKit Python package and run using Colab [28]. All merging was performed without GPU processors, as model merging is far less computationally demanding than fine-tuning. After the fine-tuning or model merging was complete, the new model was uploaded onto the HuggingFace website. All of the generated fine-tuned and merged models, as well as the datasets used in this work, can be found at: `https://huggingface.co/ihughes15234`

5.2.3 Google Cloud API for Fine-Tuned Models

Once models are stored in the HuggingFace website, they can be deployed to Google Cloud for inference. After the model is transferred from HuggingFace to Google Cloud, a custom API endpoint is created. This endpoint can then be accessed by the updated GT-Bench codebase when generating game prompts and queries for the LLMs. LLM inference parameters, such as temperature and the maximum amount of new tokens, are configured in GT-Bench and can be set uniquely for each LLM.

5.2.4 Google Cloud API for Gemini Models

For the Gemini-Pro and Gemini-Flash models, Google Cloud has API endpoints in a subset of regions which are ready for API requests. Unlike the API for fine-tuned models, the Gemini models do not require custom model and API endpoint deployments. For these models, LLM inference parameters such as temperature can still be set in GT-Bench.

Chapter 6

Evaluation Methodology and Game Parameters

In this chapter, we provide a description of *Tic-Tac-Toe*, *Prisoner's Dilemma*, and *Kuhn Poker* as these games are used in the fine-tuning and model merging experiments. Then, we discuss the evaluation metrics and methods for our implemented LLMs in GT-Bench. Finally, we provide the LLM and game parameters used in each of our experiments.

6.1 Game Details for Fine-Tuning Subset

In this section, we provide additional details for the games which are used in fine-tuning and/or model merging: *Tic-Tac-Toe*, *Prisoner's Dilemma*, and *Kuhn Poker*. Additional details on the game-play for the remaining 7 games in GT-Bench can be found in the the original GT-Bench paper. The three game descriptions below are summarized from [4].

6.1.1 *Tic-Tac-Toe*

Tic-Tac-Toe is a 2 player game on a 3x3 grid. Players take turns placing their respective symbols, X for player 1 (P_1) or O for player 2 (P_2), in each of the 9 grid spaces. Players can only play in open grid spaces. A player wins if they are able to place 3 of their respective symbol in a row vertically, horizontally, or diagonally. If there is no winner and the board is full then the game is a draw [4].

	P_1 **Silent**	P_1 **Testify**
P_2 **Silent**	P_1: -1 — P_2: -1	P_1: 0 — P_2: -3
P_2 **Testify**	P_1: -3 — P_2: 0	P_1: -2 — P_2: -2

Table 6.1: Payoff matrix for P_1 and P_2 in *Prisoner's Dilemma*

6.1.2 *Prisoner's Dilemma*

Prisoner's Dilemma is a game in which 2 players simultaneously choose whether to testify or stay silent against their opponent each round. The version implemented in GT-Bench is *Iterated Prisoner's Dilemma*, where there is a random number of rounds from 1-100 in each match. In each match, players have record of the past round decisions included in the prompt [4]. Payouts are given each round for P_1 and P_2 and are shown in Table 6.1

6.1.3 *Kuhn Poker*

Kuhn Poker is a simplified version of Poker in which one of three possible cards is dealt to each player (without replacement): King, Queen, and Jack. In the GT-Bench implementation, two players play against each other with one player going first and the other second. Both players are given one card. The first player can either pass or bet. If the first player chooses to pass and the second player bets, then the first player has the opportunity to pass or bet again. If a player's final choice is to pass while the other chooses to bet, then the betting player receives a payout of 1. If both players bet then the player with the highest card (King > Queen > Jack) has a payout of 2 while the losing player gets 0. If both players pass then each gets a score of 0 [4].

6.2 Normalized Relative Average

The Normalized Relative Average (NRA) comes directly from the original GT-Bench paper and is a measure used to normalize the scores from all games to a range of $[-1, 1]$ [4]. $NRA(P_1, P_2, f_s)$ measures the relative advantage of P_1, when competing against P_2, with the score calculation f_s. $f_s(P_1, m)$ refers to the score earned by P_1 in the m-th match ($1 \leq m \leq K$, where K is the total number of matches between the two players):

Definition 6.2.1. Normalized Relative Average (NRA) [4]

$$NRA(P_1, P_2, f_s) = \frac{\sum_m f_s(P_1, m) - \sum_m f_s(P_2, m)}{\sum_m f_s(P_1, m) + \sum_m f_s(P_2, m)} \tag{6.1}$$

For zero-sum games, e.g. *Tic-Tac-Toe*, a player in match m would receive a score of $f_s(P_1, m) = 1$ for a win, $f_s(P_1, m) = 0.5$ for a draw, and $f_s(P_1, m) = 0$ for a loss. For non-zero sum games, the score $(f_s(P_1, m))$ is equal to the rewards earned in match m. $NRA(P_1, P_2, f_s) = 0$ means that P_1 and P_2 had an even performance across K total matches. $NRA(P_1, P_2, f_s) < 0$ would show that P_2 had a higher score than P_1 across K matches. Conversely, $NRA(P_1, P_2, f_s) > 0$ shows that P_1 had a higher score than P_2 across K matches. For example, $NRA(P_1, P_2, f_s) = 1$ would tell us that P_1 had complete advantage over P_2 in K total matches.
NRA is the baseline measure which we use to benchmark our newly added LLMs. This measure is used across all ten of the games available in GT-Bench. Further, NRA is our primary metric for measuring the performance of the fine-tuned and merged LLMs in *Tic-Tac-Toe*.

6.3 Strategy Adherence

Beyond the NRA performance of our LLMs, we also test their adherence to specific strategies. Our LLMs were fine-tuned for two different strategies in two different games, *Prisoner's Dilemma* and *Kuhn Poker*.
For *Prisoner's Dilemma*, our goal is for the trained LLMs to follow the tit-for-tat strategy. We will analyze how fine-tuning impacts the adherence to the tit-for-tat strategy when compared to the base LLM model. We will also analyze strategy adherence after merging the fine-tuned *Tic-Tac-Toe* and *Prisoner's Dilemma* models.
For *Kuhn Poker*, we use the Counterfactual Regret Minimization (CFR) self-play algorithm to generate a set of answer probabilities for each possible game prompt. We then train our LLM to this mixed strategy with DPO. This DPO fine-tuned model was evaluated over 1,000 matches with the Random agent so that prompt response percentages could be compared to the CFR output. The mixed strategy for this game differs from the pure strategy in *Prisoner's Dilemma* where we have one correct move in each scenario; for *Kuhn Poker* we want to analyze if it is possible to modify next token probabilities without a pure strategy. If successful, we could generate strategically probabilistic

answers in *Kuhn Poker* or other games which rely on a mixed strategy for success.

6.4 Global Game Parameters

Unless otherwise noted, the following parameters are used for all games and players presented in our experiment results, which are given in Chapter 8:

1. 50 matches between the LLM and opponent agent.
2. Each player has 1st player advantage for half of the matches (when applicable).
3. LLM Temperature: 0.2
4. LLMs use the base prompt implementation (zero-shot) without Chain-of-Thought reasoning or multi-shot inference.
5. LLM maximum new tokens: 20

We chose a lower temperature to encourage more deterministic token generation. Early experiments with higher temperatures (1.0) led to lower completion rates and lower performance, especially for the complete and deterministic games. The ”maximum new tokens” refer to the maximum amount of allowable tokens that the LLM can generate in its response. Given that we utilized only the base or zero-shot prompting style, the LLM responses were generally 3-6 new tokens. The limit of 20 was chosen to limit any run-on text generation, which occurred infrequently but led to long run times and costly text generation.

Chapter 7

Fine-Tuning and Model Merging Implementation

In this chapter, we discuss the training approach for two LLMs, Phi 3.5 Mini Instruct and Llama 8.1 Instruct, while highlighting the datasets used for each training. For fine-tuning, two methods were used: supervised fine-tuning (SFT) and Direct Preference Optimization (DPO). Several iterations of datasets and hyper-parameters were used in the fine-tuning process. The datasets explained here and the results reported in Chapter 8 represent the most performative iterations of the fine-tunings performed. We will also disclose the percentage of samples in the training data that appeared in our post-training validation runs. After trial and error of different learning rates, a rate of 5.0e-6 was chosen for all DPO and SFT fine-tunes. For model merging, the Linear, SLERP, and Model Stock methods were used to evaluate differences between merging fine-tuned LLMs versus DPO fine-tuning on a combined game dataset. A reference table giving all of the model names, along with a brief explanation of each model, can be found in Section 7.5.

7.1 *Tic-Tac-Toe* Fine-Tuning and Dataset

For the *Tic-Tac-Toe* SFT of both Phi 3.5 Mini and Llama 8.1 Instruct, the dataset comprised of 1200 unique prompt and answer sets generated from the MCTS agent. Ten epochs, or full cycles through the dataset, were used for this fine-tuning.
Several iterations of the DPO dataset were created prior to finding success.

First, the most frequent answer per prompt was used for the "chosen" answer in the DPO dataset. This differs from the SFT dataset where each unique prompt and answer pair was used. Initially, only the non-chosen remaining **legal** moves were used as rejected responses in the DPO dataset. While this improved the *Tic-Tac-Toe* performance, this dataset formation also led to a large number of non-legal moves, and incomplete games, being generated by the fine-tuned *Tic-Tac-Toe* models. An update was then made to the rejected answer set. For each chosen answer in the DPO dataset, both the non-chosen legal and illegal moves were included as rejected answers. That is, each chosen answer now had a rejected answer entry for all other board spaces in *Tic-Tac-Toe* for a total of eight entries per unique prompt and chosen answer combination. Against the Random agent, 100 out of 354 (28.2%) prompts, or game situations, which were used in the training data were given during the Random agent versus DPO fine-tuned Phi 3.5 Mini Instruct matches.

7.2 *Prisoner's Dilemma* Fine-Tuning and Dataset

For *Prisoner's Dilemma*, a SFT was first performed using a combined dataset of 3000 samples for each of the ten games available in GT-Bench. While other games did not improve performance after this fine-tune, the *Prisoner's Dilemma* performance increased slightly. Further, the main purpose of the SFT fine-tune prior to DPO training is to have a suitable reference policy, which was achieved. The DPO training dataset comprised of 1200 unique *Prisoner's Dilemma* prompt and answer pairs from tit-for-tat agent game-play against the Random agent. This run also had 10 epochs. Against the Random agent, 229 out of 326 (70.2%) prompts, or game situations, which were used in the training data were given during the Random agent versus DPO fine-tuned Phi 3.5 Mini Instruct matches.

7.3 *Kuhn Poker* Fine-Tuning and Dataset

The initial SFT for *Kuhn Poker* also used combined dataset of 3000 samples for each of the ten games available in GT-Bench. This is the same dataset, and SFT model, mentioned in the prior section for *Prisoner's Dilemma*. While the *Kuhn Poker* performance did not increase from the SFT, it still provided a suitable reference policy for DPO fine-tuning. The DPO dataset for *Kuhn Poker* used the output from simulations run using the OpenSpiel team's CFR Python notebook [31]. The final percentage of "Pass" and "Bet" responses for each prompt was then reflected in the training dataset for 500 samples. For example, if one prompt had a response rate of 40% for "Pass" from the CFR algorithm then it would have 200 samples in the DPO dataset. The dataset had 12,000 rows in total and 5 epochs of training were performed. For this dataset, all of the 12 possible game prompts were given in the training data.

7.4 Additional Datasets and Methods Explored

Before delving into the successful results in Chapter 8, we will give an overview the datasets and fine-tuning or model merging techniques which did not lead to performance or strategy adherence improvements. Note that nearly all of the 40+ LLM fine-tunes performed for this work led to a suitable training loss. However, a majority of the fine-tuning jobs either did not increase model performance or led to low completion rates. Because of this, training loss and other training metrics were excluded from this write-up. In this case, not much information can be gleaned from training metrics as they all indicate that models were successfully fine-tuned while their GT-Bench validation runs showed a drop in performance and/or completion rates. Training metrics were primarily used to ensure that a catastrophic failure did not occur during a training, at which point the fine-tuning job would be terminated.
A subset of fine-tuning datasets that did not yield success include the following:

- 20,000 unique samples for *Breakthrough*, 1-3 epochs. SFT and DPO.
- 20,000 unique samples for both *Breakthrough* and *Connect4* (40,000 total), 1-2 epochs. SFT and DPO.
- 3,000 samples for all 10 games, 2 epochs. SFT.
- 1,200 samples for *Tic-Tac-Toe* with additional rows for the most frequent occurrences in the response database, 5 epochs. SFT and DPO.

Fine-tuning on the complex games of *Breakthrough* and *Connect4* led to a small improvement in performance that paired with a large drop in completion rates for the games. An informal analysis showed that LLMs would memorize opening moves successfully and then repeat them or make other illegal mistakes in the endgame. Training on a combined dataset for all games led to noisy and inconsistent results, paired with a near 0 completion rate in the game of *Breakthrough*. The *Tic-Tac-Toe* DPO training with additional rows for frequent moves improved performance in *Tic-Tac-Toe* but led to a lower completion rate, especially against the MCTS agent. Interestingly, this model also showed increased performance in another completion information game: *Connect4*.
Additional runs were performed with learning rates of 1.0e-4, 1.0e-6, and 5.0e-7. The 5.0e-7 and 1.0e-6 rates resulted in models which were similar in performance to the base model being trained, while the 1.0e-4 learning rate yielded models with low completion rates. None of the additional learning rates yielded a mix of performance increases and completion rate stability comparable to 5.0e-6.
For model merging, an additional run was performed using the Drop And REscale (DARE) method, which drops a random subset of parameters with the largest difference between the base and fine-tuned LLM and then rescales the remaining parameters [34]. While this method has been successful in other works, the resulting merged DARE model in this work produced a large drop in *Tic-Tac-Toe* performance and, more importantly, led to a very low completion rate.

7.5 Model Reference Table

Table 7.1 gives the names of models used for the figures in Chapter 8, with the exception of the *Kuhn Poker* models which are mentioned explicitly in the *Kuhn Poker* section. The merged models in the table were given a 50/50 split of the relevant model weights/parameters and attributes of the parent models.

Name Shown in Paper	**Base Model**	**Description**
Gemini Pro 1.5	-	As of this writing, the top tier Commercial model from Google for text inference and multi-modal applications.
Gemini Flash 1.5	-	Commercial Gemini Flash 1.5. Faster but less performant than Gemini Pro 1.5.
Llama 3.1	Llama 3.1 8b Instruct	Unsloth version of Llama 3.1 Instruct with 8 billion parameters
Phi 3.5	Phi 3.5 Instruct Mini	Unsloth version of Phi 3.5 mini instruct
Llama 3.1 TTT SFT	Llama 3.1 8b Instruct	SFT fine-tuned for *Tic-Tac-Toe* from Unsloth version of Llama 3.1 Instruct with 8 billion parameters
Llama 3.1 TTT DPO	Llama 3.1 8b Instruct	DPO fine-tuned for *Tic-Tac-Toe* from 'Llama 3.1 TTT SFT'
Phi 3.5 TTT SFT	Phi 3.5 Instruct Mini	SFT fine-tuned for *Tic-Tac-Toe* from Unsloth version of Phi 3.5 Mini Instruct
Phi 3.5 TTT DPO	Phi 3.5 Instruct Mini	DPO fine-tuned for *Tic-Tac-Toe* from 'Phi 3.5 TTT SFT'
Phi 3.5 3K SFT	Phi 3.5 Instruct Mini	SFT fine-tuned using 3000 samples from each game. Trained from Unsloth version of Phi 3.5 Mini Instruct
Phi 3.5 PD DPO	Phi 3.5 Instruct Mini	DPO fine-tuned for *Prisoner's Dilemma* from 'Phi 3.5 3K SFT'
Phi 3.5 PD and TTT DPO	Phi 3.5 Instruct Mini	DPO fine-tuned for with both *Tic-Tac-Toe* and *Prisoner's Dilemma* datasets from 'Phi 3.5 3K SFT'
Phi 3.5 Merge Linear	Phi 3.5 Instruct Mini	Linear merge of 'Phi 3.5 TTT DPO' and 'Phi 3.5 PD DPO'
Phi 3.5 Merge SLERP	Phi 3.5 Instruct Mini	SLERP merge of 'Phi 3.5 TTT DPO' and 'Phi 3.5 PD DPO'
Phi 3.5 Merge Model Stock	Phi 3.5 Instruct Mini	Model Stock merge of 'Phi 3.5 TTT DPO' and 'Phi 3.5 PD DPO'

Table 7.1: Table giving a description of each LLM shown in the figures of Chapter 8. Each model merge was done with an even (50/50) split of the relevant model weights/parameters and attributes of the parent models

Chapter 8

Results

In this chapter, the baseline results for Phi 3.5 Mini Instruct (Phi 3.5), Llama 8.1 Instruct (Llama 3.1), Gemini-Pro, and Gemini Flash are given. This baseline will first be compared to results from the original GT-Bench paper. Next, the baseline results are compared with results from the SFT and DPO fine-tuning for *Tic-Tac-Toe*, *Prisoner's Dilemma*, and *Kuhn Poker*. Finally, merged models from the *Prisoner's Dilemma* and *Tic-Tac-Toe* DPO fine-tunes will be compared to a DPO fine-tune using a combined *Prisoner's Dilemma* and *Tic-Tac-Toe* dataset.

Remark 8.0.1. All code necessary to reproduce the results is available at `https://github.com/ihughes15234/Thesis_GTBench`.

8.1 Baseline Performance and Comparison to GT-Bench

We begin with a comparison to the results from GT-Bench to check that the results from our updated codebase are reasonable. This comparison is first done using the completion rate from our games to ensure that the newly implemented LLMs can be used with a completion rate over 90%, which was the completion cutoff in the original GT-Bench paper. We also look at the average performance of our newly introduced LLMs against the Random and MCTS agents as an initial performance benchmark prior to the analysis of our fine-tuned and merged LLMs.

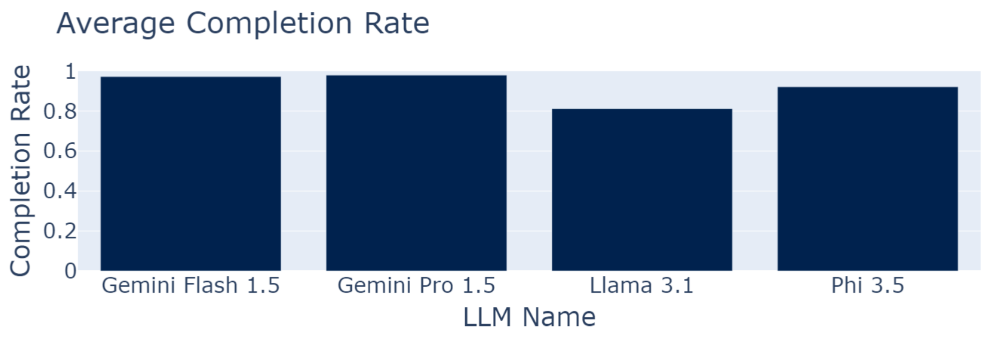


Figure 8.1: Completion rate for base LLMs

8.1.1 Completion Rate

Completion rates for all of the base model LLMs can be seen in Figure 8.1. The completion rates for Gemini-Pro, Gemini-Flash, and Phi 3.5 are all above 90%, with Phi 3.5 having overall high completion rate with the exception of the game *Breakthrough*. Given that *Breakthrough* has the highest complexity among the complete and deterministic games, this was not determined to be an issue. These results indicate that the code updates to GT-Bench to allow additional LLMs was successful. Llama 8.1 has a much lower completion rate than the rest of the LLMs and a low completion rate when compared to the LLMs in the original GT-Bench experiments. However, Llama 8.1 was used primarily to examine fine-tuning for *Tic-Tac-Toe* where it had a completion rate of 95%. Given that fine-tuning for *Tic-Tac-Toe* was the primary use of Llama 8.1, we decided to proceed with it for this purpose.

8.1.2 Average Performance of LLMs Across All Games

The average NRA performance for all of the LLMs is given in Figure 8.2. The baseline performance shows that Gemini-Pro is the superior LLM agent in terms of average NRA, followed by Gemini-Flash, Phi 3.5, and Llama 3.1. This is expected given the large number of parameters that Gemini-Pro is expected to have, with the GT-Bench results resembling ChatGPT 4 in the original GT-Bench paper. Given the small number of parameters in Phi 3.5, the performance gap is not large between Phi 3.5 and Gemini-Pro.
The performance of the four baseline LLM models against the MCTS and Random agents is similar to the performance shown in the original GT-Bench

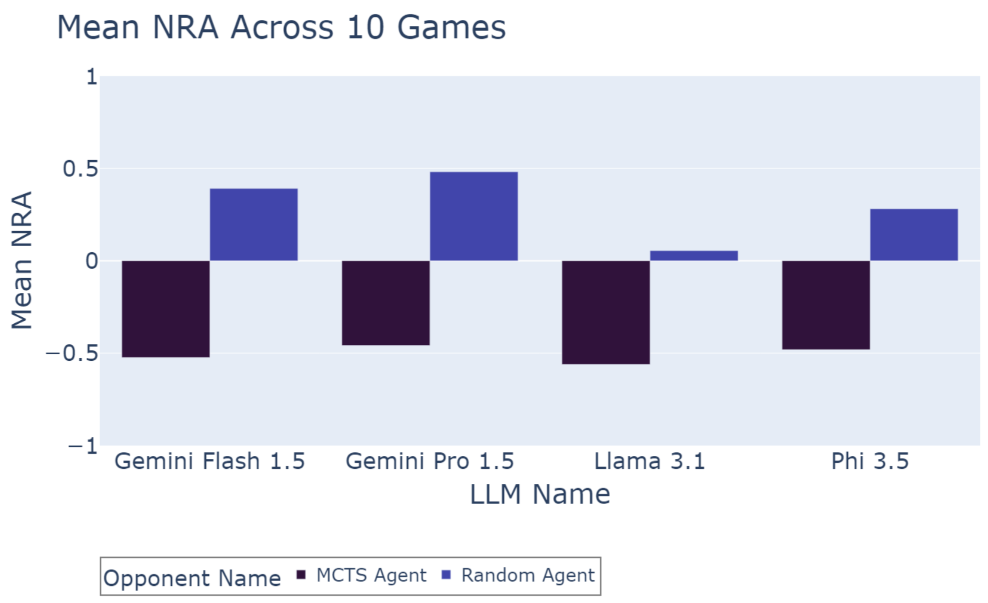


Figure 8.2: Mean NRA for all 10 games for the base models

matches. The MCTS results are given in Figure 8.3. Against the MCTS agent, the LLM models lose nearly every match in the complete and deterministic games. With the probabilistic and/or incomplete information games the LLM agents improve against the MCTS agent, but the MCTS agent still shows better overall performance. Though not shown here, there is large variation in how the LLM agents perform in each of the different probabilistic and/or incomplete information games, which matches the results in the original GT-Bench paper.

The LLMs have a higher performance than the Random agent when averaging across all games, as shown in Figure 8.4. Here there is much less of a discrepancy and pattern between the complete/deterministic games versus the probabilistic and/or incomplete information games. Llama 3.1 performs relatively poorly compared to the larger parameter agents shown in the original GT-Bench paper. The Llama 3.1 performance provides an opportunity to analyze the impact of fine-tuning on a less performative model.

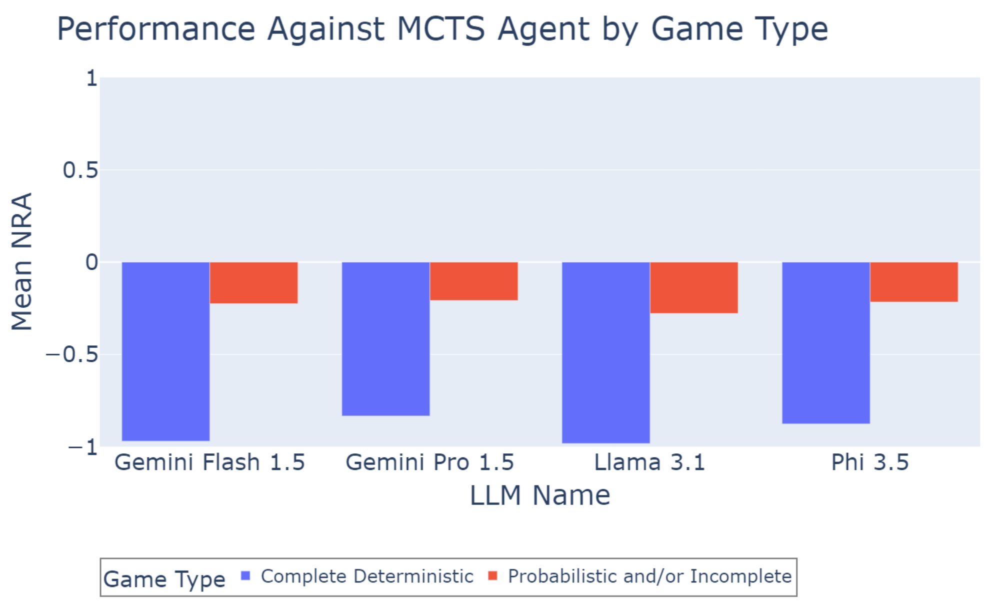


Figure 8.3: Base LLM performance against the MCTS agent, across all 10 games

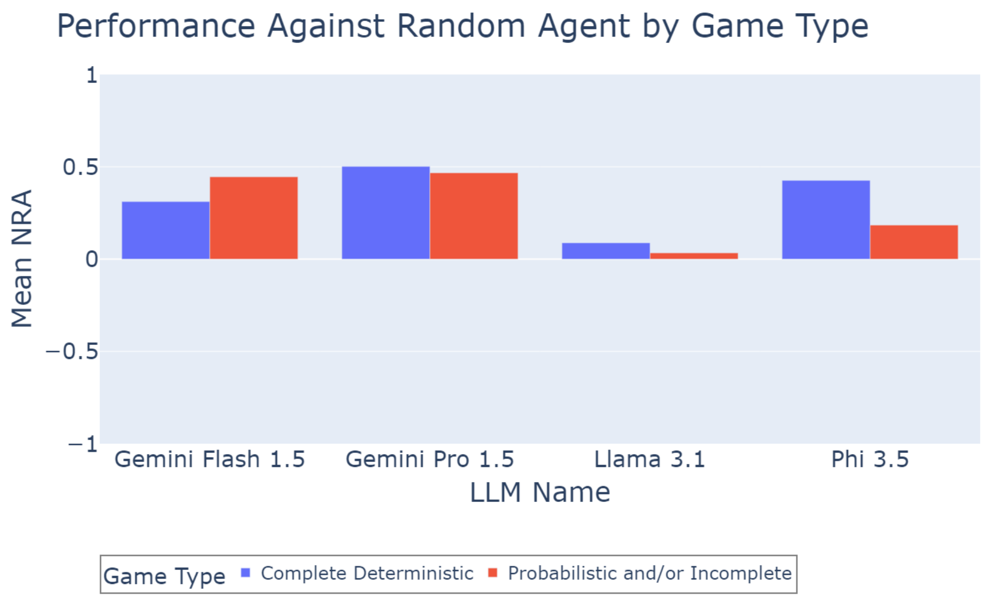


Figure 8.4: Base LLM performance against the Random agent, across all 10 games

8.2 Fine-Tuning Results

The DPO fine-tuning was successful in *Tic-Tac-Toe* for both Phi 3.5 and Llama 3.1. For *Prisoner's Dilemma* and *Kuhn Poker*, only Phi 3.5 was fine-tuned. Phi 3.5 showed a near perfect adherence to the tit-for-tat strategy after the DPO fine-tune. DPO training with Phi 3.5 for *Kuhn Poker* showed mixed success with policy adherence in 7 out of 12 scenarios. However, there was policy adherence for only 1 out of 5 scenarios with mixed strategies, showing the limitations of our ratio-based DPO training approach.

8.2.1 *Tic-Tac-Toe*

The baseline *Tic-Tac-Toe* performance for the four baseline LLMs is shown in Figure 8.5, where we see that the performance order is the same as the overall LLM performance across all games. The LLMs have an NRA ranking of Gemini-Pro, Gemini-Flash, Phi 3.5, and Llama 3.1 with none of the agents being able to win against the MCTS agent. In order to further evaluate the DPO fine-tuning results, Phi 3.5 and Llama 3.1 also had matches against both Gemini-Pro and Gemini-Flash for a total of 4 opponents for the baseline and fine-tuned models.
After the DPO fine-tuning, Llama 3.1 shows improvement against the Random agent, Gemini-Pro, and Gemini-Flash with the largest improvement being against Gemini-Flash. The DPO fine-tune results for Llama 3.1 are given in Figure 8.6. There was no improvement against the MCTS agent after the DPO training but, given the improvement against the other opponents, we can say that the DPO training was successful. The SFT fine-tuning gave a mixed but overall negative result and only improved performance against the Random agent. Though the SFT model was not successful in improving the overall performance against all of the opponents, it was still used as the reference model for DPO fine-tuning.
The DPO fine-tuning for Phi 3.5 shows improvement against all opponents, as shown in Figure 8.7. Similar to Llama 3.1, the largest improvement was against the Gemini-Flash opponent. There was a very small improvement against the MCTS agent which was the result of one tie in 50 matches. The SFT fine-tuning gave a mixed result with minor improvement against Gemini-Flash, negative results against the Random agent, and no change against Gemini-Pro or the MCTS agent. Similar to Llama 3.1, the SFT model was still used as the base reference model in the DPO fine-tuning for

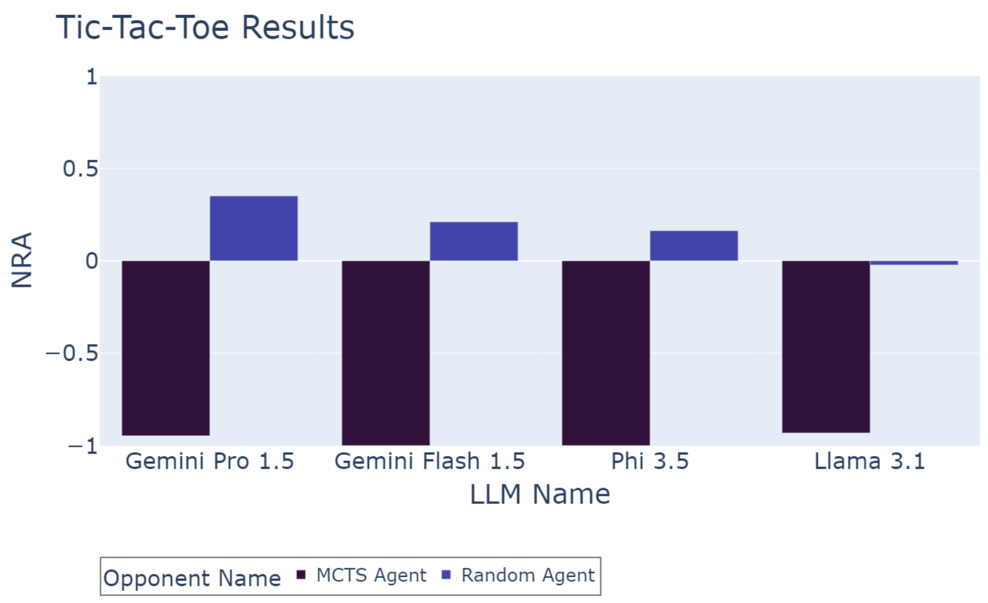


Figure 8.5: Performance of Base LLMs in *Tic-Tac-Toe*, broken out by the MCTS and Random agent opponents

Phi 3.5.
The *Tic-Tac-Toe* DPO fine-tuning showed overall success for Phi 3.5 and Llama 3.1. For both models, the training did not increase overall performance to the level of the MCTS agent but did show improvement against far larger parameter and commercial models in Gemini-Pro and Gemini-Flash.

Fine-Tuning Impact on Completion Rate and Performance

For the *Tic-Tac-Toe* DPO fine-tuned models, we also performed 50 matches for each of the ten games against the MCTS and Random agents. From these matches we can see whether the *Tic-Tac-Toe* only fine-tuning would have any impact on the other games' performance and/or the overall completion rate. As shown in Figure 8.8, the average completion rate improved for both the Llama 3.1 and Phi 3.5 models, with significant improvements for Llama 3.1. Perhaps fine-tuning with GT-Bench prompts, even those solely from *Tic-Tac-Toe*, impacted the completion rate and rule following ability for the other games. For both models, the *Breakthrough* game completion rate was hugely improved between the base and fine-tune models even without training on

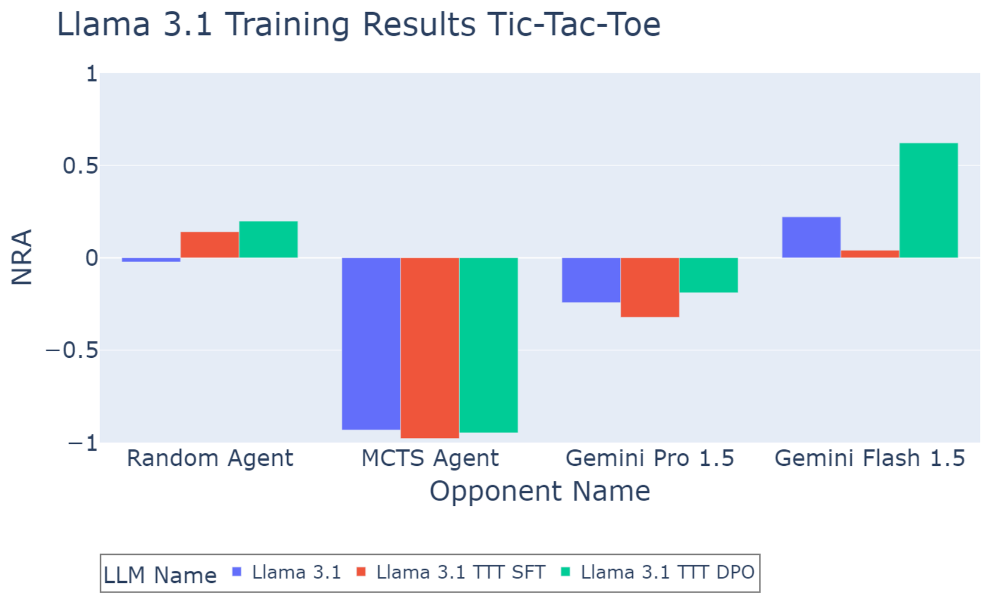


Figure 8.6: Performance of Base, SFT Fine-Tuned, and DPO Fine-Tuned Llama 3.1 8b Instruct LLM in *Tic-Tac-Toe* against the Random agent, MCTS agent, and Gemini Pro and Flash LLMs

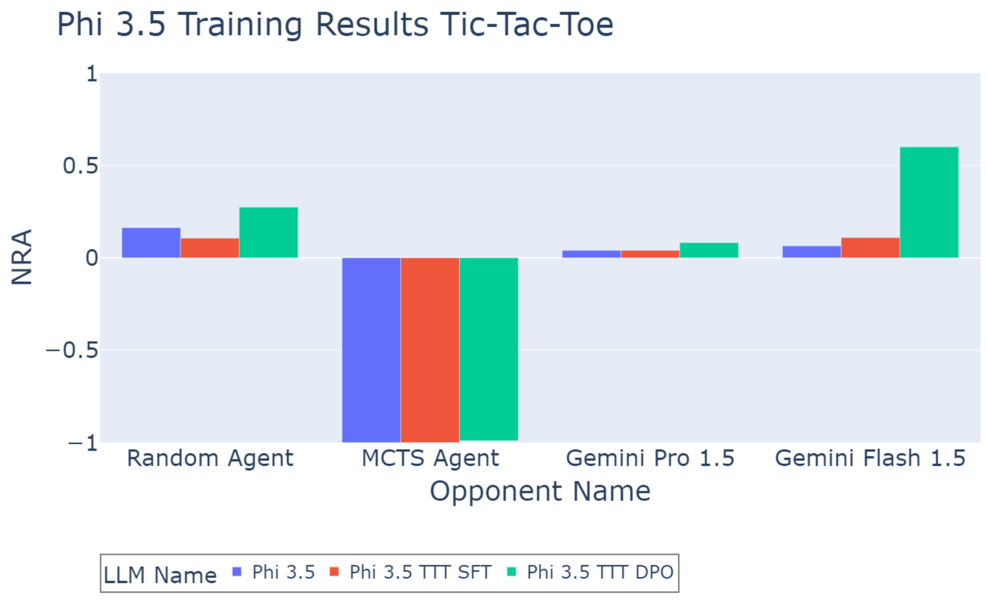


Figure 8.7: Performance of Base, SFT Fine-Tuned, and DPO Fine-Tuned Phi 3.5 Mini Instruct LLM in *Tic-Tac-Toe* against the Random agent, MCTS agent, and Gemini Pro and Flash LLMs

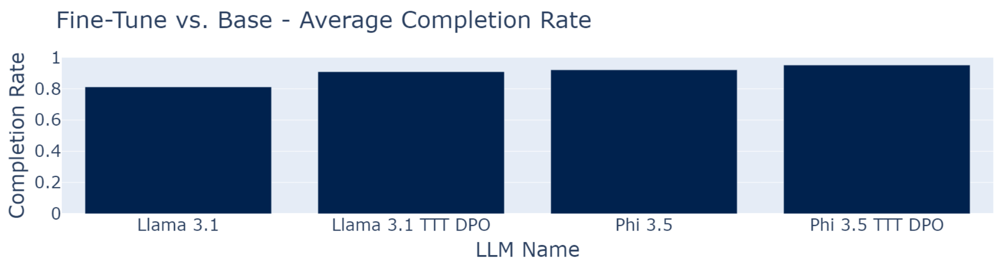


Figure 8.8: This figure gives the average completion rate across all games for both the base and *Tic-Tac-Toe* DPO fine-tuned Phi 3.5 and Llama 3.1 LLMs. Both the Phi and Llama models show improvements in overall completion rate before and after the fine-tune for only *Tic-Tac-Toe*, showing the importance of prompt formatting and instruction/rule-following in the fine-tuning process.

any of the *Breakthrough* game prompts.

The average NRA performance across all of the games (including *Tic-Tac-Toe*) shows a small improvement in performance for Llama 8.1 against the Random agent, as shown in Figure 8.9. This could indicate emergent abilities for the DPO fine-tuned Llama 8.1 *Tic-Tac-Toe* model. Conversely, against the MCTS agent there was very little difference between the base and DPO fine-tuned Llama 8.1 models. Also, the Phi 3.5 model performed worse on average against both the MCTS and Random agents. Given the conflicting results, further work focused on emergent capabilities is necessary prior to making any conclusions.

When breaking out the performance between the MCTS (Figure 8.10) and random (Figure 8.11) agents based on complete and deterministic games vs. probabilistic and/or incomplete information games we see a similar pattern as is given without the game type breakdown. Llama 8.1 has the most pronounced improvement against the Random agent for complete and deterministic games while Phi 3.5 sees a fairly consistent performance drop across all game types.

8.2.2 *Prisoner's Dilemma*

When DPO fine-tuning Phi 3.5 on the tit-for-tat strategy in *Prisoner's Dilemma*, adherence increases to nearly 97%. The SFT fine-tune shows a very moderate increase in adherence to the tit-for-tat policy. The resound-

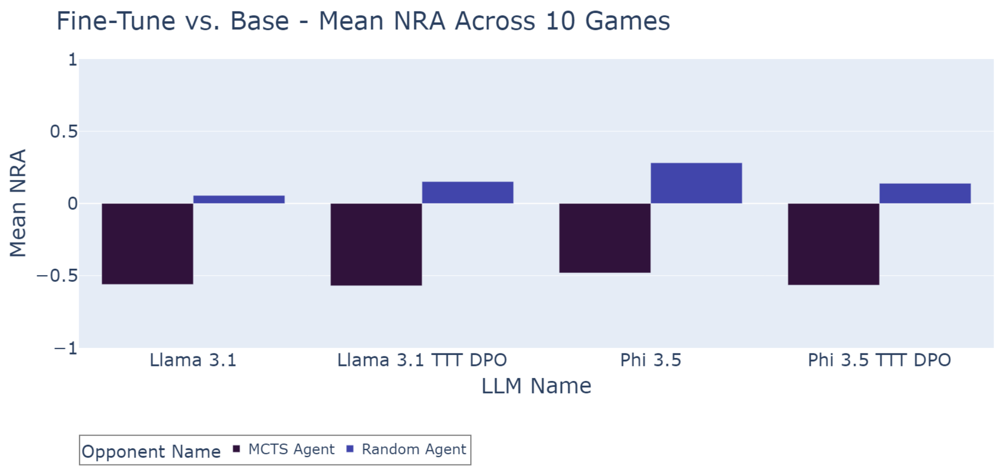


Figure 8.9: Mean NRA for the DPO fine-tuned Phi 3.5 and Llama 3.1 versus the base LLMs across all 10 games, broken out by the MCTS and Random opponents. Performance improves (on average) for Llama 3.1 but decreases overall for Phi 3.5.

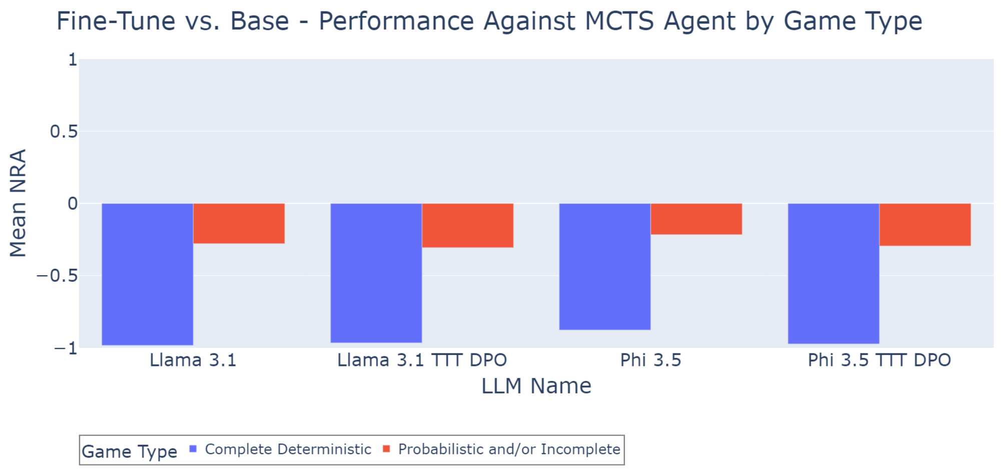


Figure 8.10: Mean NRA against the MCTS agent for the DPO fine-tuned Phi 3.5 and Llama 3.1 versus base LLMs across all 10 games, broken out by game type.

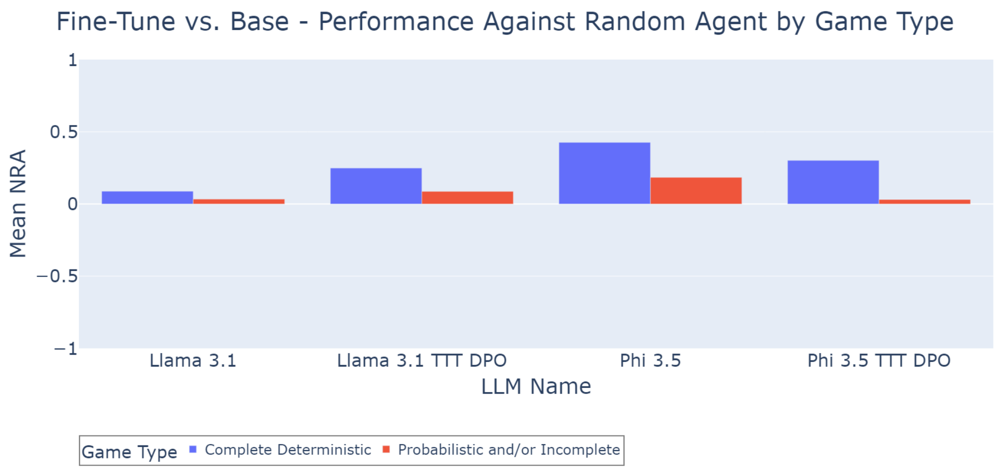


Figure 8.11: Mean NRA against the Random agent for the DPO fine-tuned Phi 3.5 and Llama 3.1 versus base LLMs across all 10 games, broken out by game type.

Model Name	**Percent Adherence**	**Moves Analyzed**
Phi 3.5	69.6%	303
Phi 3.5 3k SFT	73.6%	303
Phi 3.5 PD DPO	96.7%	276

Table 8.1: *Prisoner's Dilemma* DPO results by percent adherence to the tit-for-tat strategy

ing success of policy adherence is likely tied to the binary decision in *Prisoner's Dilemma*; the number of possible moves and game states in *Prisoner's Dilemma* is much smaller than a game like *Tic-Tac-Toe*. Note that the first round was excluded from the analyzed moves in Table 8.1 given that the tit-for-tat strategy is based on matching your opponent's prior move.

8.2.3 *Kuhn Poker*

For *Kuhn Poker*, 1000 matches against the Random agent were performed in order to build a suitable amount of samples for analysis. The results below show the impact of 5 Epochs of training with DPO, with temperatures of 0.2 (Table 8.2) and 1.0 (Table 8.3). Unlike the results in the *Tic-Tac-Toe* and *Prisoner's Dilemma* sections, the DPO fine-tune for *Kuhn Poker* gave mixed

	CFR Strategy		**Phi Fine-Tune**		**Phi Base**	
Game State/LLM Prompt	Pass	Bet	Pass	Bet	Pass	Bet
King	37.9%	62.1%	0.0%	100.0%	0.0%	100.0%
King Opponent Bet	0.0%	100.0%	0.0%	100.0%	0.0%	100.0%
King Opponent Pass	0.0%	100.0%	0.0%	100.0%	0.0%	100.0%
King Pass, Opponent Bet	0.0%	100.0%	-	-	-	-
Queen	100.0%	0.0%	100.0%	0.0%	0.0%	100.0%
Queen Opponent Bet	66.2%	33.8%	2.6%	97.4%	0.0%	100.0%
Queen Opponent Pass	100.0%	0.0%	100.0%	0.0%	0.0%	100.0%
Queen Pass, Opponent Bet	45.4%	54.6%	31.6%	68.4%	-	-
Jack	79.3%	20.7%	100.0%	0.0%	98.1%	1.9%
Jack Opponent Bet	100.0%	0.0%	100.0%	0.0%	89.9%	10.1%
Jack Opponent Pass	66.7%	33.3%	100.0%	0.0%	43.5%	56.5%
Jack Pass, Opponent Bet	100.0%	0.0%	100.0%	0.0%	89.5%	10.5%

Table 8.2: Temperature 0.2 DPO 5 Epoch

success. There was policy adherence in 7 out of 12 scenarios but adherence for only 1 out of 5 scenarios with mixed strategies. Namely, the 'Queen Pass, Opponent Pass' scenario shows adherence after a temperature of 1.0. All other mixed scenarios were unsuccessful with many tending towards a binary response closer to the dominant strategy for that category. The notable exception to this was 'Queen Opponent bet' where the fine-tuned model actually tended towards the non-dominant strategy response. Experiments were also run with 3 and 7 Epochs, as well as tweaked CFR data distributions. In all other scenarios, there was no adherence to the mixed strategies from the CFR algorithm, with only a subset of the pure strategies being successful for these runs.

The 'Queen Pass, Opponent Pass' game state is the mixed strategy with the closest distribution to 50% for each response. This could indicate that a mixed strategy response can best be adhered to with DPO when a distribution is close to 50%. In the other scenarios, responses quickly diverge from the desired answer distribution given from the CFR algorithm. While DPO was successful for one mixed strategy, these experiments did not lead to any repeatable training procedure to successfully adhere to mixed strategy responses.

	CFR Strategy		Phi Fine-Tune		Phi Base	
Game State/LLM Prompt	Pass	Bet	Pass	Bet	Pass	Bet
King	37.9%	62.1%	0.0%	100.0%	0.0%	100.0%
King Opponent Bet	0.0%	100.0%	0.0%	100.0%	0.0%	100.0%
King Opponent Pass	0.0%	100.0%	0.0%	100.0%	0.0%	100.0%
King Pass, Opponent Bet	0.0%	100.0%	-	-	-	-
Queen	100.0%	0.0%	100.0%	0.0%	0.0%	100.0%
Queen Opponent Bet	66.2%	33.8%	2.5%	97.5%	0.0%	100.0%
Queen Opponent Pass	100.0%	0.0%	100.0%	0.0%	0.0%	100.0%
Queen Pass, Opponent Bet	45.4%	54.6%	42.3%	57.7%	-	-
Jack	79.3%	20.7%	100.0%	0.0%	98.1%	1.9%
Jack Opponent Bet	100.0%	0.0%	100.0%	0.0%	89.9%	10.1%
Jack Opponent Pass	66.7%	33.3%	100.0%	0.0%	43.5%	56.5%
Jack Pass, Opponent Bet	100.0%	0.0%	100.0%	0.0%	89.5%	10.5%

Table 8.3: Temperature 1.0 DPO 5 Epoch

8.2.4 Combined *Prisoner's Dilemma* and *Tic-Tac-Toe* Fine-Tune

For comparison with the merged models, a DPO fine-tune was performed with a combined dataset from the *Prisoner's Dilemma* and *Tic-Tac-Toe* DPO fine-tunes. This resulted in the 'Phi 3.5 PD and TTT DPO' model. For *Tic-Tac-Toe* performance, matches were performed against the Random agent, MCTS agent, Gemini-Pro, and Gemini-Flash. For *Prisoner's Dilemma*, matches were performed against the Random agent to test adherence.

8.3 Model Merging Results

While model merging was successful to varying degrees for *Prisoner's Dilemma*, there was no success for any of the merged models with *Tic-Tac-Toe*. The combined DPO fine-tune showed success for *Prisoner's Dilemma* and showed mixed results in *Tic-Tac-Toe* play with improvement compared to the base model against three out of four opponents. The mixed results for the combined DPO model suggest that fine-tuning for strategic games becomes less effective when the dataset scope increases. These results could also be sensitive to the complexity of *Tic-Tac-Toe* and further work is needed to understand why the combined fine-tune and model merging was less successful for

Model Name	**Percent Adherence**	**Moves Analyzed**
Phi 3.5	69.6%	303
Phi 3.5 3k SFT	73.6%	303
Phi 3.5 PD DPO	96.7%	276
Phi 3.5 PD and TTT DPO	97.8%	276
Phi 3.5 Merge Linear	82.2%	444
Phi 3.5 Merge SLERP	89.9%	286
Phi 3.5 Merge Model Stock	70.3%	444

Table 8.4: Model merge results by percent adherence to tit-for-tat strategy

Tic-Tac-Toe relative to *Prisoner's Dilemma*.

8.3.1 *Prisoner's Dilemma*

Results for all of the Phi 3.5 LLMs are given in Table 8.4. The 'Phi 3.5 PD and TTT DPO' model had the best adherence to the tit-for-tat strategy. Surprisingly, the adherence was very close to 'Phi 3.5 PD DPO' which was trained solely with *Prisoner's Dilemma* data. The merged models showed different levels of adherence with 'Phi 3.5 Merge Model Stock' showing next to no improvement in adherence while 'Phi 3.5 Merge SLERP' had a near 90% adherence rate. This adherence rate gives 'Phi 3.5 Merge SLERP' a performance closer to 'Phi 3.5 PD DPO' than the base model, 'Phi 3.5'. Given that model merging requires substantially less compute than a DPO fine-tune, this shows one instance where, depending on the combination of models being merged, the compute to performance trade-off could work in favor of model merging over an additional combined dataset DPO or SFT fine-tune.

8.3.2 *Tic-Tac-Toe*

None of the merged models showed NRA score improvement in *Tic-Tac-Toe*, with all of the model merges giving a decrease in performance against a majority of opponents when compared to the base and DPO fine-tuned Phi 3.5 models. The results are shown in Figure 8.12. The combined DPO model, 'Phi 3.5 PD and TTT DPO', showed improvements over the base model against the Random agent, MCTS agent, and Gemini-Flash. Against Gemini-Pro, 'Phi 3.5 PD and TTT DPO' gave the worst performance across

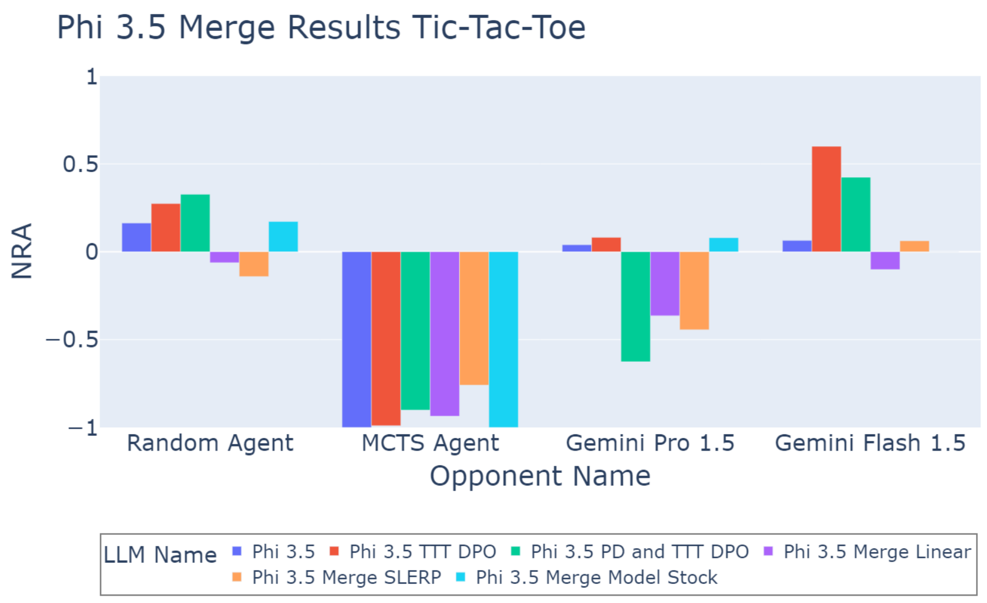


Figure 8.12: Phi 3.5 Merged Model Results for *Tic-Tac-Toe*. *Note: Against Gemini-Flash, 'Phi 3.5 Merge Model Stock' had a score of 0 so it does not show visually in the graph.*

all models shown. *Tic-Tac-Toe* does not give straightforward improvement results like *Prisoner's Dilemma* and a different training strategy may be needed for combined dataset fine-tuning and model merging in games with a large decision space like *Tic-Tac-Toe*.

Chapter 9

Discussion

In this work, we showed the impact of fine-tuning both a small (Phi 3.5) and small/medium (Llama 8.1) size LLMs with SFT and DPO. Further, we presented the differences in effectiveness between several model merging techniques and performing a larger, combined game dataset DPO training. We demonstrated that small, open-source LLMs can improve play in *Tic-Tac-Toe* with DPO fine-tuning. This same training technique increased strategy adherence in the incomplete information game of *Prisoner's Dilemma*. It is unclear whether fine-tuning with DPO can adequately train a small LLM to play a mixed strategy for a game such as *Kuhn Poker*, but other fine-tuning methods should be evaluated before ruling this out. Model merging showed promise for the simpler task of strategy adherence in *Prisoner's Dilemma*, but did not show improvement in the more complex task of *Tic-Tac-Toe*. The results from this work show that further research is needed in the area of strategic games, especially with respect to the emergent capabilities of LLMs with fine-tuning. That is, there is not enough evidence from this work to rule out emergent capabilities from the fine-tuning process, especially given the small scale with which fine-tuning took place for these experiments. This same research could also be repeated with larger LLMs and over additional training epochs to see if model parameter size has a significant impact on the fine-tuning or emergent capabilities of the LLMs.
Additional work on the project could add more prompting styles to GT-Bench, such as multi-agent debate, where 2 or more LLMs can debate until they reach a consensus pick for the appropriate move in a game [35]. This could help generate additional training data while also having the potential to save on processing costs if multiple small LLMs can outperform one larger

LLM. Additional training datasets could use strategy or reasoning from online sources or textbooks instead of specific move outputs to evaluate how LLMs adapt their response without direct training data. Combining any of these techniques with Chain-of-Thought prompting could also help identify issues in the LLMs reasoning steps which need to be improved.

Bibliography

[1] M. Wortsman, G. Ilharco, S. Y. Gadre, *et al.*, *Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time*, 2022. arXiv: `2203.05482 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2203.05482`.

[2] K. Shoemake, "Animating rotation with quaternion curves," *SIGGRAPH Comput. Graph.*, vol. 19, no. 3, pp. 245–254, Jul. 1985, ISSN: 0097-8930. DOI: `10.1145/325165.325242`. [Online]. Available: `https://doi.org/10.1145/325165.325242`.

[3] D.-H. Jang, S. Yun, and D. Han, *Model stock: All we need is just a few fine-tuned models*, 2024. arXiv: `2403.19522 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2403.19522`.

[4] J. Duan, R. Zhang, J. Diffenderfer, *et al.*, *Gtbench: Uncovering the strategic reasoning limitations of llms via game-theoretic evaluations*, 2024. arXiv: `2402.12348 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2402.12348`.

[5] H. Naveed, A. U. Khan, S. Qiu, *et al.*, *A comprehensive overview of large language models*, 2024. arXiv: `2307.06435 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2307.06435`.

[6] M. Peeperkorn, T. Kouwenhoven, D. Brown, and A. Jordanous, *Is temperature the creativity parameter of large language models?* 2024. arXiv: `2405.00492 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2405.00492`.

[7] V. B. Parthasarathy, A. Zafar, A. Khan, and A. Shahid, *The ultimate guide to fine-tuning llms from basics to breakthroughs: An exhaustive review of technologies, research, best practices, applied research challenges and opportunities*, 2024. arXiv: `2408.13296 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2408.13296`.

[8] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, *Direct preference optimization: Your language model is secretly a reward model*, 2024. arXiv: 2305.18290 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2305.18290.

[9] E. Yang, L. Shen, G. Guo, *et al.*, *Model merging in llms, mllms, and beyond: Methods, theories, applications and opportunities*, 2024. arXiv: 2408.07666 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2408.07666.

[10] J. Wei, X. Wang, D. Schuurmans, *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, Curran Associates, Inc., 2022, pp. 24 824–24 837. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

[11] S. Minaee, T. Mikolov, N. Nikzad, *et al.*, *Large language models: A survey*, 2024. arXiv: 2402.06196 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2402.06196.

[12] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27, Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf.

[13] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[14] L. Ouyang, J. Wu, X. Jiang, *et al.*, "Training language models to follow instructions with human feedback," in *Advances in Neural Information Processing Systems*, 2022. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf.

[15] J. Huang and K. C.-C. Chang, *Towards reasoning in large language models: A survey*, 2023. arXiv: `2212.10403 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2212.10403`.

[16] M. F. A. R. D. T. (FAIR), A. Bakhtin, N. Brown, *et al.*, "Human-level play in the game of diplomacy by combining language models with strategic reasoning," *Science*, vol. 378, no. 6624, pp. 1067–1074, 2022. DOI: `10.1126/science.ade9097`. eprint: `https://www.science.org/doi/pdf/10.1126/science.ade9097`. [Online]. Available: `https://www.science.org/doi/abs/10.1126/science.ade9097`.

[17] E. Akata, L. Schulz, J. Coda-Forno, S. J. Oh, M. Bethge, and E. Schulz, *Playing repeated games with large language models*, 2023. arXiv: `2305.16867 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2305.16867`.

[18] C. F. Tsai, X. Zhou, S. S. Liu, J. Li, M. Yu, and H. Mei, *Can large language models play text games well? current state-of-the-art and open questions*, 2023. arXiv: `2304.02868 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2304.02868`.

[19] K. Gandhi, D. Sadigh, and N. D. Goodman, *Strategic reasoning with language models*, 2023. arXiv: `2305.19165 [cs.AI]`. [Online]. Available: `https://arxiv.org/abs/2305.19165`.

[20] X. Feng, Y. Luo, Z. Wang, *et al.*, *Chessgpt: Bridging policy learning and language modeling*, 2023. arXiv: `2306.09200 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2306.09200`.

[21] A. Ruoss, G. Deletang, S. Medapati, *et al.*, "Amortized planning with large-scale transformers: A case study on chess," in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. [Online]. Available: `https://openreview.net/forum?id=XlpipUGygX`.

[22] M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk, "Monte carlo tree search: A review of recent modifications and applications," *Artificial Intelligence Review*, vol. 56, no. 3, pp. 2497–2562, Jul. 2022, ISSN: 1573-7462. DOI: `10.1007/s10462-022-10228-y`. [Online]. Available: `http://dx.doi.org/10.1007/s10462-022-10228-y`.

[23] H. Touvron, T. Lavril, G. Izacard, *et al.*, *Llama: Open and efficient foundation language models*, 2023. arXiv: `2302.13971 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2302.13971`.

[24] M. Abdin, J. Aneja, H. Awadalla, *et al.*, *Phi-3 technical report: A highly capable language model locally on your phone*, 2024. arXiv: `2404.14219 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2404.14219`.

[25] G. Team, *Gemini: A family of highly capable multimodal models*, 2024. arXiv: `2312.11805 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2312.11805`.

[26] E. J. Hu, Y. Shen, P. Wallis, *et al.*, *Lora: Low-rank adaptation of large language models*, 2021. arXiv: `2106.09685 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2106.09685`.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[28] C. Goddard, S. Siriwardhana, M. Ehghaghi, *et al.*, *Arcee's mergekit: A toolkit for merging large language models*, 2024. arXiv: `2403.13257 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2403.13257`.

[29] M. Lanctot, E. Lockhart, J.-B. Lespiau, *et al.*, "OpenSpiel: A framework for reinforcement learning in games," *CoRR*, vol. abs/1908.09453, 2019. arXiv: `1908.09453 [cs.LG]`. [Online]. Available: `http://arxiv.org/abs/1908.09453`.

[30] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione, "Regret minimization in games with incomplete information," *Advances in neural information processing systems*, vol. 20, 2007.

[31] E. Lockhart and O. Team, *Motivation, core api, implementing cfr and reinforce on kuhn poker, leduc poker, and goofspiel*, 2020. [Online]. Available: `https://github.com/google-deepmind/open_spiel/blob/master/open_spiel/colabs/CFR_and_REINFORCE.ipynb`.

[32] M. H. Daniel Han and U. team, *Unsloth*, 2023. [Online]. Available: `http://github.com/unslothai/unsloth`.

[33] G. Research, *Colaboratory*, `https://research.google.com/colaboratory`, Accessed: 2024-07.

[34] L. Yu, B. Yu, H. Yu, F. Huang, and Y. Li, *Language models are super mario: Absorbing abilities from homologous models as a free lunch*, 2024. arXiv: `2311.03099 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2311.03099`.

[35] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch, *Improving factuality and reasoning in language models through multiagent debate*, 2023. arXiv: 2305.14325 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2305.14325.