Teaching Large Language Models to play games

Jeff Jiang

Project Advisor: Constantinos Daskalakis

1 Introduction

Recent Large Language Models exhibit an interesting general range of capabilities. Networks trained on the traditional "next-word" prediction task are geting increasingly better, even superhuman, at general daily tasks that people engage in. With such a promising realm of developments, many are beginning to speculate whether this could lead us to something truly interesting, and debate is ongoing about whether or not "next-token prediction" as the underlying training method, coupled with transformers with the self-attention mechanisms, will continue to improve its performance and become a source of general intelligence.

However, more and more research and discussion have been focusing on what the current LLMs, even the most advanced and best performing model like GPT4, can't achieve. In class, we talked about some of the current limitations in LLMs, notably its limited ability in math and the prone tendencies of LLMs to hallucinate. We note in this paper an additional limitation of GPT on logical reasoning skills, even struggling to play the most simple game of TicTacToe, which is played at expert-level by many kids in kindergarten. This lack of expertise in this simple game highlights the LLM's inability as of yet to think ahead like humans in reasoning when playing games that require strategic thinking. [1]

One obvious factor that leads to this poor performance of GPT4 is the lack of gameplay data in its training set. This makes intuitive sense, as we do not reasonably expect the training data behind GPT4 to contain boardgame configurations of TicTacToe and possible strategies and values for each case - with the training data mainly coming from internet content sources, the training space is much richer, but much less targeted and specific than well-performinng agents trained by AlphaZero-like procedure. This will be the main focus of this paper's effort, and we'll integrate LLM training with the self-play and Monte Carlo Tree Search based methods used in AlphaZero (Silver et al. (2017)), using the policy and value targets in the MCTS to finetune the target LLM. We also proposed an alternative approach to the same problem, training an auxillary network that takes as input the embeddings output by a LLM, then through a simpler network output the desired value and policy predictions. With such a variation, we circumvent the earlier problem with large token-space, computational complexity, and makes the problem of optimization easier.

After showing that it's possible for a LLM to play a game reasonably well through the training methods above, a more interesting problem would be: how much is the learned skills pure imitation of behaviors, and how much is it some latent logical reasoning capabilities. Taking a step back, is there **any** logical thinking capacities of LLM that allow it to internalize some of the MCTS-like abilities to look ahead learned from training? These problems are hard and difficult to answer and requires a much more in-depth and extensive research beyond this project, but we offer some perspectives based on the results we have, and outline the possible sequence of steps beyond of this project that will answer this problem in a more principled manner. [2]

1.1 Contributions

The contributions of the paper are as follows: we experimented and proposed zero-shot prompting strategies that improves a LLM's performance on game playing. We also combined self-play and MCTS of AlphaZero and fine-tuning of LLM to improve the model to play games well, achieving a better strategic behavior. Since this method is computationally expensive and difficult to optimize, especially for models with billions of parameters like LLaMA, we also propose an alternative approach that leverages an auxillary network to improve LLM's performance at gameplay. Lastly, we evaluate the resulting model and outline the future steps at investigating the generalizability of these results and the potential for LLM to acquire logical thinking abilities that makes it a general good player.

[1]Some might attribute this as a lack of transformers architecture on spatial thinking skills, not having a convolutional layers like some other Deep Learning architectures. However, GPT4 has been able to perform well on other tasks involving spatial reasoning skills (and we mentioned the example of it learning to draw pictures when given spatial contexts). In addition, there are wide range of examples in Li et al. (2023) and AranKomat (2018) where a transformers architecture is used to obtain positive results of good performing agents.

[2]The code for this project is available at https://github.com/jeffjiang1204/cs229projsubmission.git

1.2 Related Works

The recent developments of LLM is propelled by the transformers architecture (Vaswani et al. (2017)), which can capture long-term dependencies and exploit sequential structure in the input embeddings using the self-attention mechanism. Recently, advances in GPT4 (OpenAI (2023)) demonstrated its capabilities at dealing with various generic downstream tasks, even showing signs of disrupting the labor market through their abilities of substituting human labor in various fields (Eloundou et al. (2023)). The model weights of GPT4 and GPT3.5 is not open sourced, but meta's open source LLaMA model (Touvron et al. (2023)) has been able to perform well on multiple evaluation metrics and is used for experimentation in this project. Since even the smallest LLaMA model is large and doesn't fit well into a standard optimizational hardware, we also consider training another large-scale transformer-based language model: GPT2 (Radford et al. (2019)). GPT2 has the advantage of being more computationally accessible, while simultaneously having nearly all the components of a LLM and the framework can be easily substituted with LLaMA or other models provided we have access to better computational capacity.

On the front of learning to play games well, AlphaZero (Silver et al. (2017)), the improved variant of AlphaGo (Silver et al. (2016)) that relies solely on self-play and MCTS without integrating prior knowledge and heuristics, has demonstrated its capacity of super-human play in the game of Go and Chess. In this paper, we adopt their strategy of self-play and MCTS to train a language model.

To the best of my knowledge, not a lot of literature has covered the topics of using language models (or transformers) to play games. Li et al. (2023) studied the problem of teaching a GPT-like model to play Othello, but we note that the paper focus on training a transformer from scratch instead of fine-tuning an existing model, having different objectives than we do for this paper. Tsai et al. (2023) studies LLM's ability to play other text games, but we note that the kind of games we're considering is different, and that we're considering the traditional game types like TicTacToe, Othello, Checkers, Chess and Go, where two agents take turn making moves on a board, which encapstulate the goal of reasoning ahead adversarially and using this search to choose an optimal move.

2 Zero-shot game playing and prompting with GPT4

We begin with the observation that GPT4, even though good at many downstream tasks, is not good at playing the simplest strategic game: TicTacToe. An example is given in Figure 1. In the example on the left, the optimal move that results in a direct win will be to play action A2. However, GPT4 selects another move (notably B2). A similar query to ChatGPT (GPT3.5) finds that it does not even learn to identify areas in the board that's valid to play (the action it proposes is invalid).

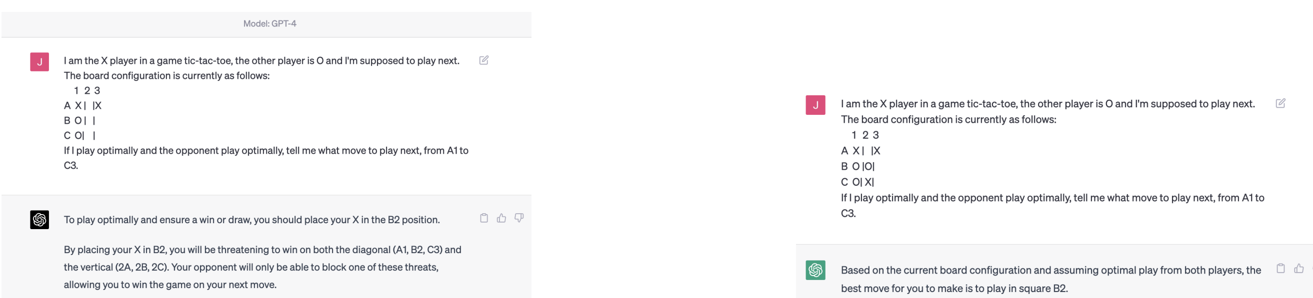

Figure 1: Playing TicTacToe with GPT4 and GPT3.5

One might look at the above query and wonder whether it's because text-based model and inputs don't align well with these games that require spatial reasoning. However, the left plot in Figure 1 actually shows that GPT4 is able to parse the graph and what the next move looks like, it's simply that it doesn't know how to play well.

2.1 Improving game-play ability with Zero-shot prompting

Our first task is to seek to improve gameplay abilities of GPT4 (and GPT3.5) with zero-shot prompting. Intuitively, the task of TicTacToe should be simple enough that a language model that can handle a lot of other harder tasks should be able to do well. To enable it to play well, however, we need to teach it to look ahead and discover which move yields potentially the highest future payoff. Furthermore, we want the description to be more general and not specific to TicTacToe (simply referencing 'you're playing the game of TicTacToe') so that we can potentially switch the description to a different game and get better results.

We first start with GPT3.5, which during testing turns out to be extremely prone to hallucinations and difficult to prompt. It turns out that the following prompting works best according to the more than 50 prompts I've tried:

```
   Here's the description of a game:  The game is played on a grid that's 3 squares by 3 squares.  You are X, your
friend is O . Players take turns putting their marks in empty squares.  The first player to get 3 of her marks in a
row (up, down, across, or diagonally) is the winner.  When all 9 squares are full, the game is over.  If no player
has 3 marks in a row, the game ends in a tie.  We label the grid from A1 (top left) to C3 (bottom right).  Only a single
pawn can be on any position, so you can't play where there's already a pawn.  In the board, there are currently 4 pawns.
Specifically, B1 is occupied by O, A3 is occupied by O, C1 is occupied by X, C3 is occupied by X. All other positions
are valid, therefore the valid positions for you to play is A1, A2, B2, B3, C2.  Player X (or O) win if a following
combination, each in any ordering, exist:  A1, A2, A3; or B1, B2, B3; or C1, C2, C3; or A1, B1, C1; or A2, B2, C3;
or A3 ,B3 ,C3; or A1, B2, C3, or A3, B2, C1.  Player X plays next.  Consider all moves that's valid for player X to
play.  For each possible move you simulated, now there's an additional X pawn on the board in the position of your
move, list the X pawns and the O pawns on the board.  There should be 3 X pawns (including C1 and C3), and 2 O pawns
(at B1 and A3).  If among the X pawns you listed on the board, that are either one of A1, A2, A3; B1, B2, B3; C1, C2,
C3; A1, B1, C1; A2, B2, C3; A3 ,B3 ,C3; A1, B2, C3; or A3, B2, C1, each in any order, X wins.  For each case tell me
whether X wins and the winning combination that has to be already on the board as an X pawn according to what you listed.
Choose the move that leads to a win.  Give your answers in text purely, don't visualize the board.
```

Using the prompt above, among 20 randomized runs in GPT3.5 (new session window), it gets right about half of the time, identifying that C2 will lead to a direct win (although among the cases where it gets right, it sometimes misidentify other actions that could falsely lead to a win. Overall the probability of it recommending a unique optimal answer that is correct is 1/5 in the 20 runs tested).

With the more advanced GPT4, we can teach it to think forward and analyze the possible steps that the opponent will do. The example used in above for GPT3.5 prompting is one where X has a direct chance of winning. What about an example where you need to think one step ahead in preventing an opponent from winning? Turns out this is possible and an example is given in Figure 2. This figure also give a nice example on how to teach language models to think ahead and prepare for various opponent strategits when making a decision, which is not really feasible in GPT3.5 due to its tendencies to hallucinate in practice, but for GPT4 and future models with higher accuracy, may give us a good search heuristic.

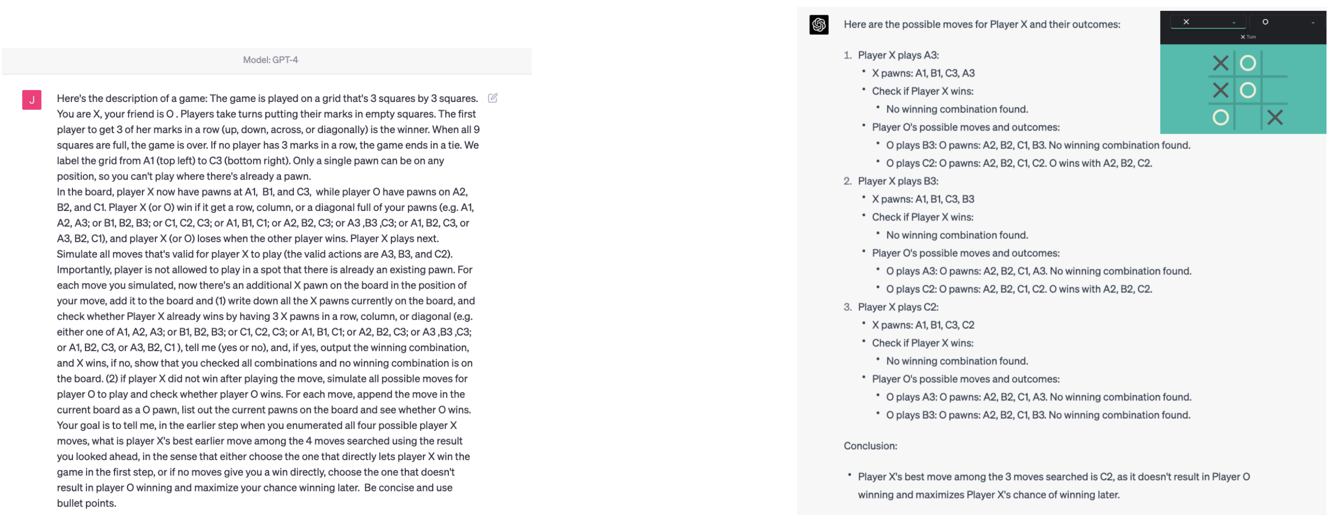

Figure 2: Playing TicTacToe with GPT4 with prompting

The above looks a bit promising as a zero-shot prompting technique, especially with the case of GPT4, but we note that this approach don't seem to generalize well to more complex game beyond TicTacToe. For one, complex games have much larger game trees and brute-forcing searching even two or three steps might be very complex. In addition, with no clear end of game configurations in sight, there is no metric of evaluating how different moves perform. These are the problems that can be solved with MCTS, and while it's hard to teach GPT4 to run MCTS, we can leverage it as part of training, which we turn to next.

3 Improving LLM performance through MCTS

3.1 Methodology

The idea of using MCTS in LLM is simple: since a LLM is itself a transformer, a deep learning architecture, we can directly train it using AlphaZero style training. However, a few things are different therefore we outline our key innovation in terms of methodology:

3.1.1 Approach 1: Extracting policy and value estimates from the LLM

We start with the observation that we can directly query a LLM for a policy or value estimate. However, traditional training of transformers and language model in particular is based on the model of next token prediction. Yet, getting an output in the form of a tokenized string that decodes to "The policy estimate based on your provide configuration is given by ..." clearly do not fit into the paradigm of the training pipeline. Therefore, we need a way to extract gradients from the model output that is different from pure generation, which we'll need in order to update the model.

In order to solve this problem, we first note that we can simplify the task by asking the language model to output as concisely as possible, adding the keyword of "No explanations" in GPT4 or LLaMA. With this, the model suppresses the output space to only the query you asked.[3] This does not yet fully solve our problem, and we'll need to consider the case of value and policy estimates differently.

- Extracting differentiable policy estimate:

  For the prompt, we will describe the game and board and ask the language model for an action. An example prompt used in training is the following: `"[Omitted description of Rules of the TicTacToe game]. We label the grid from 1 (top left) to 9 (bottom right). The current board is configured as [insert string representation of board]. If I play optimally, the next play should be at position numbered"`.

  With this prompt, the model will return a single token (encoding 1 to 9) representing the action that the actions that it thinks the agent should take. We can therefore query the logits on the token space and extract the tokens specifically corresponding to 1 to 9, which will be the logits corresponding to the policy estimate of the model. This has the nice interpretation that we're still outputing a probability distribution over all the possible actions.

  Note that the above setup is particular to the game of TicTacToe. In general, we can label the game space as a combination of characters and numbers: in the game of Othello, for example, where the game is 8×8, we can represent the game as from A1 to H8, and substitute that in the game space description above. In some tokenizers, output like A1 will be encoded into two different tokens. In this case, we can still compute the logits using the following algorithm:

Algorithm 1 Gradient extraction algorithm for policy estimates

```
/* Using the example of Othello, with an 8 by 8 board: */
mask0 ← indices of tokens corresponding to alphabetical characters A to H;
mask1 ← indices of tokens corresponding to 1-8;
/* Get the probability of first character: */
part1 ← softmax(outputlogits[-1])[mask0];
for i ← 0 to 7 do
    Sample the language model with an additional token corresponding to integer i + 1;
    part2 ← softmax(newoutputlogits[-1])[mask1];
    probability_i ← part1[i] × part2;
end
return probability
```

This algorithm essentially calculates the joint probability of the desired actions and we combine them to form the desired probabilities.

- Extracting value estimate:

  The process for extracting value estimates is similar to the above but with some small but important variation. We consider two potential methodology:

[3] With GPT2, we can simply look at the immediate few tokens it outputs

1. The first methodology follows closely with what we have in Algorithm 1 above. Instead of asking for the next action, we ask instead for a direct estimate of the probability of the player of winning (`What is the probability of me winning?`). We then compute the value estimate differentiably by the following:

Algorithm 2 Gradient extraction algorithm for value estimates

```
/* Using the example of Othello, with an 8 by 8 board:                        */
mask ← indices of tokens corresponding to 0-9;
value ← [0,1,2,3,4,5,6,7,8,9]
Output ← Joint probability that the first character decodes to 0, followed by a character that decodes to '.';
/* We want to ensure that the output has form 0.xxx                           */
Penalize the objective by c×(1-Output);
Add tokenized '0' and '.' to existing list of tokens;
for i ← 0 to 3 do
    Sample the language model with the list of tokens;
    output + = softmax(newoutputlogits[-1])[mask1]*value;
    Sample a token from the probability distribution then add it to the list of tokens;
end
/* Converts the expected value output to a number between -1 and 1            */
output = output·2 − 1
return output
```

The above algorithm is essentially simply calculating an expected value over the value estimates, using the logits output by the model as probability. We notice that this is nontrivially difficult, and involves additional penalization to the objective that might make the optimization process hard.

2. The second methodology therefore adopts a simpler approach. Instead of asking the language model to "estimate the probability of me winning", we can simply let it to "`give a number from 1 to 9 representing the likelihood of me winning, with 1 being most likely and 9 being least likely`". With this simplified approach, we know that the output will be just 1 token and thus we can extract the value by simply multiplying the probabiities of 1 to 9 with a vector denoting the discretized likelihood, e.g. $[-1, -0.8, -0.5, -0.2, 0, 0.2, 0.5, 0.8, 1]$. This gives us an easier way of getting the expected value estimates, which is not perfect (not continuous for example) but much simpler than the earlier approach.

With us extracting differentiable policy and value estimates, we can directly apply the loss function and MCTS process as in AlphaZero and train the language model.

3.1.2 Approach 2: Creating auxillary network for value and policy estimation

We note that the above problem comes with 2 major drawbacks. First of all, the optimization process might not be entirely smooth: for the training set, we're constraining ourselves to only focus on 36 (26 alphabetical characters and 10 numbers) of the $\geq 32,000$ tokens in the tokenization space. This both potentially makes our optimization problem harder, and potentially disrupts the original pretrained LLM. Intuitively, since AlphaZero relies on millions of iterations of training, with millions of rounds of training each stepping towards putting probability mass on numerical and alphabetical characters, the model might lose its origianl capabilities. We will also examine whether this is the case next section when we discuss the empirical results. Secondly, modern LLMs are extremely large. The recently released Llama model comes with different parameter sizes, and the smallest of all, the 7B version, takes 26G of GPU memory just out of parameter count. In practice, when performing the operations above, the highest GPU capacity I can get from AWS ($\approx$ 45G GPU memory) still gives me a CUDA memory error. Therefore, for Approach 1 I ended up training GPT2 instead, which puts less strain on GPU memory but works similarly to any other LLM[4].

Instead of training the model directly, we can train an auxillary network instead, which takes as input the embeddings output by a LLM (for example Llama), and output policy and value estimates. This solves the above problem, giving us an action and policy estimate that's both problem specific, as well as demanding less requirement on GPU memory (we can run inference using CPU, or with less precision sizes). An visualization of the auxillary network is given below in Figure 3.

[4] GPT2 is used primarily for sentence completion rather than Llama and GPT4 which directly helps with downstream task and can answer questions as illustrated above. We note however that for our use case, these two are similar, and we can let GPT2 complete the sentence to get the estimate, e.g. 'If I play optimally my next action will be"

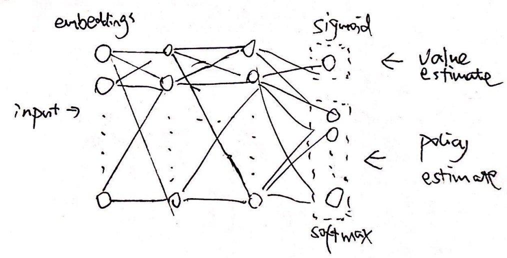

Figure 3: Auxillary network illustration

3.2 Experimental setup

We conducted experiments on both approaches above. For approach 1, we finetuned GPT2-medium but the methodology is general and can be applied to any LLM (with sufficient compute). For approach 2, we extracted embeddings from LLaMA by directly using the last hidden layer (there are definitely other approaches to getting an embeddings, and a lot of online discussions are on what is optimal, e.g. taking a sum of the last three hidden layers. Here, we simply extract the last hidden layer due to simplicity). For training both tasks, we used the g5.12xlarge instance in AWS, with 4 GPUs. For both approaches, we focus on training for the game TicTacToe. We note that both the framework and the code easily extends to more complex games. However, with inference and MCTS steps > 10 times slower due to the LLM model size compared to standard AlphaZero-like training, which already takes days to train Othello and Chess, we defer investigation to these complex games to further research beyond this course.

3.3 Experimental Results

3.3.1 Degeneracy of the game TicTacToe

The game TicTacToe is, probably familiar to all readers, very unfair (between player that move first and second) and easily lead to draws. In my first attempts to train it, both approaches seem to converge in about a day of training with a reasonable loss. When querying specific choices, however, the algorithm's decision don't make much sense. It turns out that the algorithm learns to fully converge to a solution where, during self-play and using the policy estimates, the game will result in a draw nearly 100% of the time, which results in a near-zero value loss since you can always predict 0 for value estimation. When I perform self-play using the derived strategies for 100 games, the game results in a draw 100% of the time. This is even with a positive exploration factor c as mentioned in the algorithm of AlphaZero (Silver et al. (2017)).

To fix this degenracy, I added dirichlet noise to the prediction, with a factor $\alpha = 1$ and $r = 0.75$. Formally, denote p as the output policy of the network, the actual prior passed to MCTS is given by:

$$p' = r \cdot p + (1 - r)Dir(1)$$

3.3.2 Final results

After fixing as above, I'm able to achieve reasonable performance for both approach 1 and approach 2. For evaluation, we start with Table 1, which presents the probability of the model recommending a valid move given all possible board configurations in TicTacToe. It is worthwhile to note that for both approach 1 and 2, the cases for recommending invalid action upon inspection nearly always happen when the board is almost completely filled and there's no probability for winning nor losing. Still, this suggest rooms for improvement and further training might help (we train for less than 1 day due to time and resource limitations). Table 2 below summarizes the performances of Approach 1 and Approach 2 and original GPT2 model against a random strategy player as well as the performance of Approach 1 (fine-tuned GPT2 model) against the original GPT2 model. Figure 3 documents the number of invalid moves made during a 2000-game simulation, and Figure 4 provides a few specific illustrations of the decisions of the trained model.

We see that overall the learned model is performing much better than the random player and learns to pick the correct optimal move the majority of times. Table 2 shows that the trained agent performs generally well against

Approach	Valid Moves
Approach 1	4908 out of 5000
Approach 2	4714 out of 5000
Random Player	1842 out of 5000
Original GPT2	2122 out of 5000

Table 1: Probability of playing a valid move for three approaches.

	Approach 1			Approach 2			Original GPT2			Approach 1 vs GPT2		
1000 Games	**Win**	**Draw**	**Lose**	**Win**	**Draw**	**Lose**	**Win**	**Draw**	**Lose**	**Win**	**Draw**	**Lose**
Move First	910	82	8	937	41	22	797	30	173	1000	0	0
Move Second	631	317	52	598	190	212	504	34	462	1000	0	0

Table 2: Performance results for two approaches against a random player.

a random agent and superb (2000-0 win) against the original model. This is partly due to the original GPT2 model being quite deterministic regardless of what boardstate it gets in inputs, outputing the same probability distribution over actions, as well as being a byproduct of the self-play training method, where agents starts with the original model and learns to play optimally against itself. From the two incorrect examples in Figure 4, we see that the trained model is more likely to realize when there's a row that could lead to a win or lose, and less likely to do so with veritical columns or diagonal wins and loses (although it still gets it correct sometimes, as seen in the lower middle example). This is a bit surprising since the transformer architecture supposedly should be able to handle this, being able to capture the context and relationships between different words and phrases beyond just those of immediate neighbors. We note here that it might be the case that signals that far away items are still relatively hard to associate, and require longer training time to fully capture. Still, the suboptimal plays made doesn't result in a direct lose, and however the other agent acts in turn, there is still an optimal action that will result in a win. Therefore, this inability to recognize the winning move directly might be a byproduct of training, where we don't penalize according to how long the game is played, and as long as the game results in a win, a reward of 1 is offered.

3.3.3 Evaluating the learned model

As stated above, we're also curious about whether or not the large iterations in training will degrade the original model, since all the target in the potentially tens of thousands of iterations in training include only tokens that decode to numerical numbers and/or alphabetical characters. For GPT2 which we used, there isn't a particular easy metric to use as benchmark, other than seeing whether the sentence completion make sense. In this spirit, we present some example dialogues below in Table 4.

A quick view of the results show that the trained model loses some but not all of its original abilities, suggesting that the damage with pure AlphaZero like training might not be as damaging as it appears. A word of caution though would be that training for a game like Othello and Chess will entail much longer and more iterations. Whether or not the model with many times more training iterations will further degrade in performance on

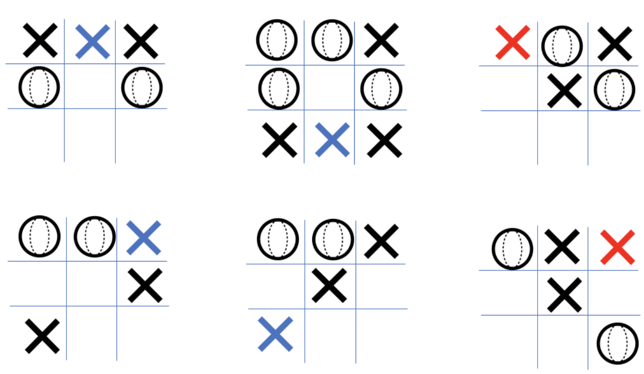

Figure 4: Example plays of the trained model (Blue denotes making the optimal move, Red denotes suboptimal move)

Approach	Total invalid moves during 2000 games simulation
Approach 1	17
Approach 2	148
Original GPT2	16768

Table 3: Probability of playing a valid move for three approaches.

Prompt	Trained model	Original model
a b c d	a b c d e f g h	a b c d e f g h
Canada is a country in the	Canada is a country in the middle of a sea	Canada is a country in the midst of a massive housing crisis. The average price of a detached home...
Boston is a city in	Boston is a city in the middle of a river	Boston is a city in which the police are not allowed to be involved in the criminal justice system.
Hello, I'm a language model	Hello, I'm a language modeler.\n\n\nCocky, I'm a language	Hello, I'm a language modeler. I'm a language modeler.

Table 4: Examples of original prompts, model outputs, and original outputs.

traditional task will be an interesting future research question.

Lastly, we note again that our approach is generalizable to any game framework, and can be very easily substituted to more complex games like Othello and Chess. Due to the short timeframe of the project, we don't have time to investigate these more complex games, but I'll continue the research beyond the course.

3.3.4 Generalizability

We briefly tested whether or not a simple variation in prompt semantics will result in learned model acting accordingly. That is, if we change the prompt from "`The first player to get 3 of her marks in a row (up, down, across, or diagonally) is the winner`" to "`The first player to get 3 of her marks in a row (up, down, across, or diagonally) loses`", whether or not the learned strategies will change. Upon experimentation, the policy estimates in Figure 4 stays the same. Therefore, a naive approach of trying to make LLM learn generic reasoning skills that are game agnostic is not applicable here. Additional strategies and ideas will be the focus on subsequent extension of this research beyond the course and are outlined next.

3.3.5 Future Direction

A future direction of this would be to expand beyond the game of TicTacToe, where we can better understand the generalizability of learned models across games (e.g. whether a fine-tuned model on playing checkers will play Othello modestly well), and investigate training methodologies and even variations in the language model architectures itself that would enable the LLM to learn to think sequentially and reason ahead in a similar way that AlphaZero can do through MCTS. An additional interesting problem is taking a step back and asking whether a language model possesses 'any' strategic thinking ability at all. There are some hypothetical ways of testing this hypothesis, and an approach I currently have in mind is to generate different forms of games each with different rules and teach expert-level play to the language model. Another approach, suggested by Professor Daskalakis, is to directly teach language model to find the Nash Equilibrium of a normal form game, or simply, to focus on the zero-sum game, where the goal is to teach the language model to think about Maxmin/Minmax strategies, thus engaging in logical reasoning. We'll pursue these directions in subsequent work.

4 Conclusion

With this project, we illustrated that it is possible to combine MCTS and self-play in AlphaZero with Large Language Model fine-tuning, and presented experimental results on the performance of GPT2 and the auxillary model (used on Llama's embeddings) on the game TicTacToe. We also outline the next steps that will help us answer the broader research problem (can we teach LLM to reason and think ahead) and we'll attempt to continue this line of work beyond this course project to answer this question.

References

AranKomat. 2018. AlphaZero Transformers. https://github.com/AranKomat/Alpha-Transformer

Tyna Eloundou, Sam Manning, Pamela Mishkin, and Daniel Rock. 2023. GPTs are GPTs: An Early Look at the Labor Market Impact Potential of Large Language Models. arXiv:2303.10130 [econ.GN]

Kenneth Li, Aspen K Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. 2023. Emergent World Representations: Exploring a Sequence Model Trained on a Synthetic Task. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=DeG07_TcZvT

OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815* (2017).

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

Chen Feng Tsai, Xiaochen Zhou, Sierra S Liu, Jing Li, Mo Yu, and Hongyuan Mei. 2023. Can Large Language Models Play Text Games Well? Current State-of-the-Art and Open Questions. *arXiv preprint arXiv:2304.02868* (2023).

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).