# Optimizing Large Language Models: Learning from Mistakes in Gameplay

Federica D'Alvano Kirakidis, Lily Gao, Aaron George, Alex Huang, Niv Levy
Mentors: Prof. Benjamin Von Roy, Yifan Zhu, Henry Widjaja

## Abstract

In recent years, there has been a surge in research and public interest in Large Language Models (LLMs), which have demonstrated remarkable potential across diverse applications and domains. This paper provides a comprehensive survey of the applications of LLMs, particularly focusing on their roles and capabilities within multi-agent systems (MAS). We utilized Gemini 1.5 Flash by Google to introduce a benchmark for evaluating LLM learning based on mistakes in previous data. Our findings reveal significant variations in LLM performance across different prompt engineering strategies, enhancing our understanding of their strategic thinking in relation to learning through game data. Additionally, we explore the complexities of extending LLM-based self-supervised learning to MAS, emphasizing coordination and communication among agents. By identifying underexplored areas and promising research directions, this survey lays the groundwork for innovative research at the intersection of LLMs, game logic, and MAS, advancing toward Artificial General Intelligence (AGI).

## Background

Natural Language Processing (NLP) has revolutionized how machines understand and interact with human language. At the core of NLP advancements, Large Language Models (LLMs) are designed to process and generate human-like text. These models—such as GPT-4, BERT, and Gemini—can perform numerous text-based tasks through training with large data samples, ranging from translation to summarization.

The training process of NLP models involves exposing them to extensive datasets containing diverse examples of natural human language. During this process, models learn patterns, grammatical rules, and contextual relationships between words and phrases. However, to further enhance the accuracy of these models, various techniques are used:

1. **Adversarial Attacks:** One common method to improve NLP models is through adversarial attacks. These attacks involve slightly perturbing the input data in a way that is imperceptible to humans but can lead to significant errors in model predictions. By training models to withstand adversarial examples, researchers enhance their resilience and reliability. This approach is crucial in applications where accuracy and robustness are

critical, such as financial models or autonomous systems. The Fast Gradient Sign Method (FGSM) (Figure 1) is a popular technique that leverages the gradient of the loss function. Then, it quantifies the amount of pixel values for a given image that must be changed to prevent the language model from accurately identifying it. This process is demonstrated in Figure 2 (TensorFlow).

$$adv\_x = x + \epsilon * \text{sign}(\nabla_x J(\theta, x, y))$$

*Figure 1*

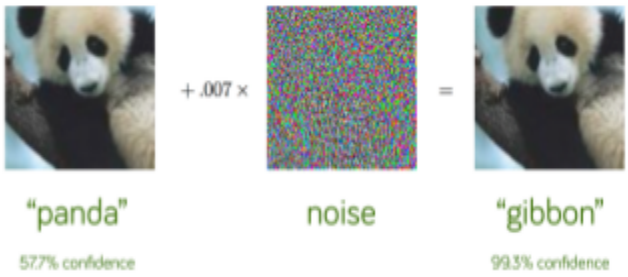


*Figure 2 (TensorFlow)*

2. **Data Augmentation:** Another technique for NLP improvement involves augmenting training data with variations of existing examples, helping models improve generalization and reliability with unseen data. Techniques like synonym replacement, random insertion, and paraphrasing are commonly used to create augmented datasets.
3. **Regularization Methods:** Regularization techniques like dropout, weight decay, and batch normalization are employed to prevent overfitting, ensuring that the model performs well on both training and test datasets.

While traditional training methods focus on enhancing model architecture and training processes, our research explores the optimization of LLMs through **prompt engineering.**

Prompt engineering involves crafting precise and effective prompts that guide the model to produce desired outputs. By optimizing these prompts, we aim to enhance the model's performance on specific tasks without altering the underlying model architecture (Tong, Yongqi, et al., 2024). LLMs primarily operate with string inputs, limiting their ability to utilize traditional methods like gradient-based adversarial attacks directly on non-text data. Techniques such as FGSM and other adversarial strategies, while effective in highlighting vulnerabilities in image models, are not directly applicable to LLMs. Thus, strategies like prompt engineering aim to optimize LLM performance and robustness within the constraints of their text-based input

nature. We emphasize prompt engineering with various ways of instructing past data or omitting this data entirely.

# Methods

We devised four methods to test Tic Tac Toe gameplay optimization through prompt engineering. All methods are based on the same structure of code: the "Agent," or non-optimal player, plays first ("X") by calling upon the Google Gemini API to decide on moves (Google). Then, the "Optimal Agent" ("O"), which is hard-coded to play the optimal move, plays opposite the Agent. The Optimal Agent always either wins or forces a tie. For the fourth method, however, the Agent plays against a "Near-Optimal Agent," which chooses a random optimal move instead of the first optimal move.

| **Play 1: Loss** | **Optimal Moves: 2/3** |
|---|---|
| Order: 4,6,5 |  |
| Observations: failed to block | |
| **Play 2: Loss** | **Optimal Moves: 3/4** |
| Order: 4, 7, 2,5 |  |
| Observations: blocked once effectively, but failed to block a second time | |

*Figure 3:* Observations and optimal move %

*Figure 4: Data collection for one "trial"*

We tested every method in a series of five trials. Each trial was made up of ten games. For every game, we recorded the final board, the order of plays, and the number of self-judged optimal moves to better test the efficiency of the prompting (Figure 3, 4). We noted whether Agent gameplay improved between rounds through not only statistical measurement but also through

human observation—Was the agent making the same mistakes or correcting them? Did it exhibit patterns in optimal or non-optimal gameplay? Did it address specific issues tackled in prompts?

## Basic Game Logistics & Optimal Agent:

Each of the nine spaces on the game board was assigned a number zero through eight to easily define the grid state (occupied or unoccupied) in an array, as seen in Figure 5 (RealPython). The Optimal Agent was hard-coded to play 100% optimally and then proven optimal through fifty plays against a human (Figure 6). After deeming the Optimal Agent truly optimal, we played it against the Agent in the first three methods and collected data.

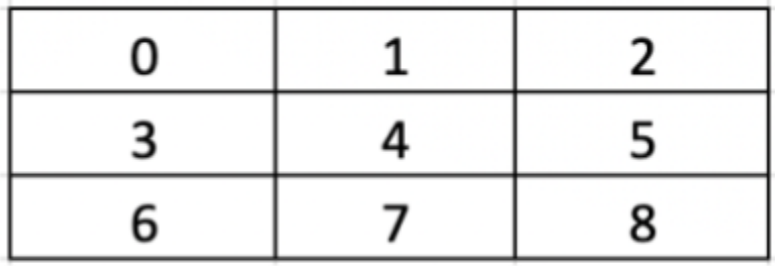


*Figure 5 - Board grid format*

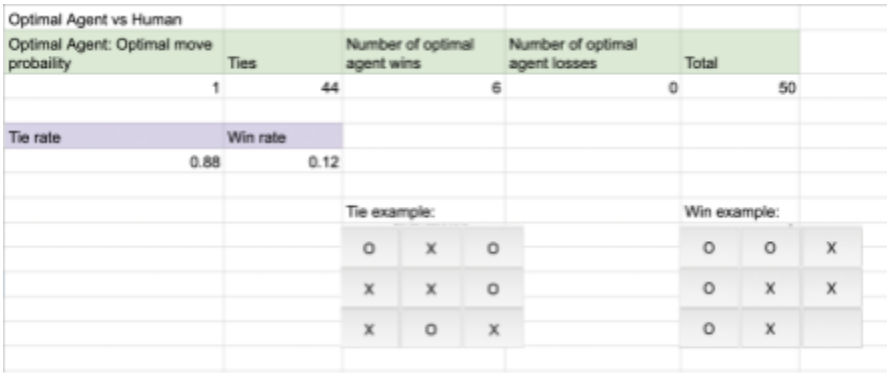


*Figure 6 - Proving Optimal Agent to be Optimal*

## Base Prompt:

To ensure an equal starting point for all methods, we developed a base prompt to build upon. This prompt includes the game state (showing the occupied and unoccupied grids), the instructions for Tic Tac Toe (including a description of the game format and the different ways the Agent can lose), the role of the Agent as a player, and the valid JSON response format and invalid response formats (Topsakal et al).

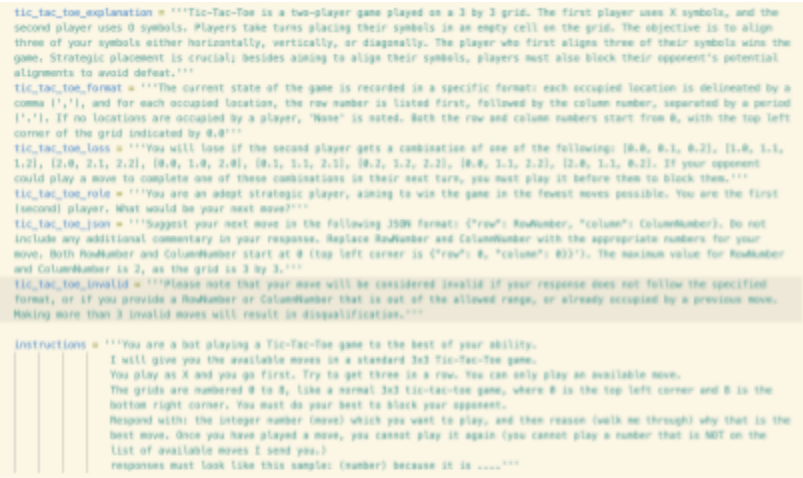


*Figure 7 - Base Prompt*

A preliminary trial with the basic prompt was gathered to act as a baseline to which the others could be compared; this baseline resulted in a 100% loss rate for the Agent (Figure 8).

| Prob. of Agent | Prob. of Optimal | # Agent Wins | # Agent Losses | # Ties | Total Games |
|---|---|---|---|---|---|
| 0 | 0.394 | 0 | 10 | 0 | 10 |

*Figure 8* - baseline performance for coded Optimal Agent

## Method 1: Complexity

In the first method, we observed the relationship between the level of detail (character count) in the Tic-Tac-Toe game instructions without game data and the improvement in Agent gameplay. Added detail did not include explicit references to past gameplay, but rather general elaboration on the Tic-Tac-Toe strategy (Nicholls). For example:

*"Try to get three in a row **diagonally, horizontally, or vertically. Block the other player from getting three in a row horizontally, diagonally, or vertically. When choosing a move, you must consider offensive strategies (trying to win) or defensive strategies (trying to block the other player).** (...)"*

## Method 2: Past Game Data - List Form

In the second method, we observed how listing past game data in the LLM prompt correlated with improved Agent gameplay. The following instructions accompanied the game data: "Utilize the following sequence of events from past games to develop a winning strategy, anticipate the

opponent's next move, block, strategize your moves, force ties, and prevent losses." Game data from the previous trial was collected and listed, resulting in ten data sets to be analyzed. Data lists would purposefully include losses and ties to test whether the Agent could employ winning strategies and avoid repeating mistakes.

The game data was written in a numbered list with some commentary on the move's effect, which winner won, and the specific grids within the winning line. (e.g. "X played grid 1, blocking a row" ). (Figure 9)

```
Game 6:
      1. X played grid 4
      2. O played grid 0
      3. X played grid 7
      4. O played grid 1 blocking a potential column
      5. X played grid 2 blocking a potential row
      6. O played grid 3
      7. X plays grid 5
      8. O plays grid 6 winning the game with the column [0.0, 1.0, 2.0]
```

*Figure 9*

### Method 3: Past Game Data - Long-Form

In the third method, we followed the structure of Method 2, but placed game data in paragraph-long instructions instead of structured lists, observing whether detailed data explanations would correlate to optimized gameplay. For example:

*"Game 1: You lost the game. First, you played grid 4. Your opponent played grid 0. You played grid 7. The opponent played grid 1. You played grid 3 and failed to block your opponent from winning three-in-a-row. Your opponent played grid 2, winning a row with a combination [0.0, 0.1, 0.2]."*

### Method 4: Near-Optimal Agent

In the fourth method, we followed the structure of Method 2 but played the Agent against a Near-Optimal Agent. The Near-Optimal Agent selected a random optimal move instead of the first choice of an optimal move, thus breaking away from the predictable patterns that the Optimal Agent often displayed.

# Results

All methods were tested through a series of five trials and a baseline trial with no prompt engineering (Methods, Base Prompt). Each trial included ten Tic-Tac-Toe plays. For each trial within a method, the following data was collected based on the ten plays: "Probability of Agent

Tie", "Probability of Optimal Move", "Number of Agent Wins", "Number of Agent Losses", and "Number of Agent Ties". For the fourth method, an additional "Probability of Agent Win" was calculated.

The *Probability of Agent Tie* is based on the number of ties per trial between the Agent and the Optimal Agent.

The *Probability of Optimal Move* is calculated based on the number of optimal moves and the total number of moves made by the Agent per trial. A move is deemed optimal when the first play is the center grid, the move blocks the Optimal Agent from winning, or the move opens up the possibility of a three-in-a-row. If the move fails to meet the criteria, it is not deemed optimal.

The game boards from each play and the order of moves for each play were recorded, totaling fifty boards per method to be analyzed for trends and used as data for prompts.

For Methods 1, 2, and 3, the *Probability of Agent Tie* was the deciding factor in whether the Agent improved between each game. Since the Agent played against the Optimal Agent, a tie would indicate that the Agent reached the same level of optimal play as its opposite. For Method 4, however, the *Probability of Agent Win* was the primary measure of optimal play because the Agent played the Non-Optimal Agent and could win against it.

## METHOD 1

Adding only to the complexity and number of sentences in the Agent instructions produced minimal to no improvements in gameplay. The *Probability of Agent Tie* did not correlate with the increase in character count for each successive trial. However, the *Probability of Optimal Move* increased slightly across the five trials (Figure 10).


*Figure 10*

Often, the Agent and the Optimal Agent played the same series of moves, resulting in dozens of repeated game boards (Figure 11). Sometimes, the Agent was able to force a tie. Thus, the number of Agent Wins, Agent Losses, and Agent Ties did not clearly increase or decrease with increased sentence complexity across the trials (Figure 12).


*Figure 11 - Boards are color-coded by similarity*

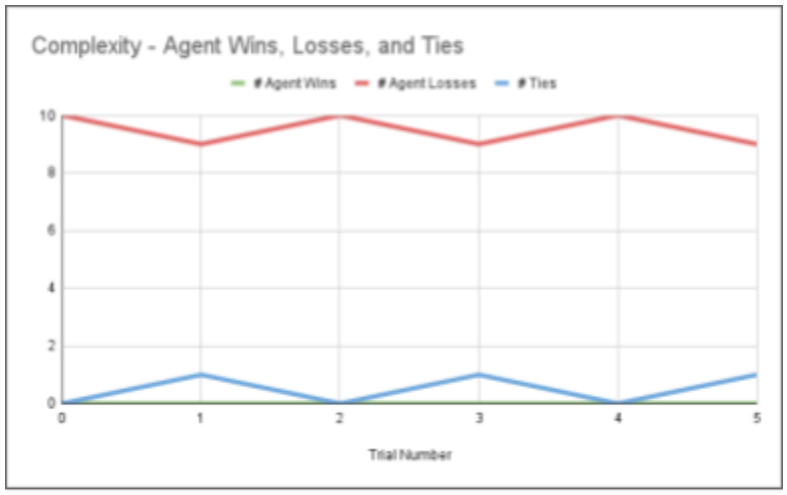


*Figure 12*

Increasing only the complexity (character count) of prompts for each trial likely did not improve gameplay because the prompts explained strategies without reference to previous mistakes. Furthermore, repeated game boards likely occurred because both the Agent and the Optimal Agent resorted to familiar patterns of optimal play. Thus, the Agent likely did not "learn" from previous trials, but instead received more general instruction. Overall, the use of this method did not correlate with Agent optimization.

## METHOD 2

Listing game data from the previous trial in the Agent instructions produced strong improvements in gameplay, leading to optimal Agent gameplay in Trial 5. The *Probability of Agent Tie* increased with each game trial, reaching 1. The *Probability of Optimal Move* also increased with each game trial and reached 1 (Figure 13).


*Figure 13*

Initially, the Agent and the Optimal Agent played many of the same series of moves, resulting in dozens of repeated game boards (Figure 14). Over time, however, patterns of ties were repeated until the Agent and the Optimal Agent only played ties in the last trial. The number of *Agent Wins* remained at 0; the number of *Agent Ties* increased and reached 1; the number of *Agent Losses* decreased and reached 0 (Figure 15). By Trial 5, the Agent played 100% optimally.

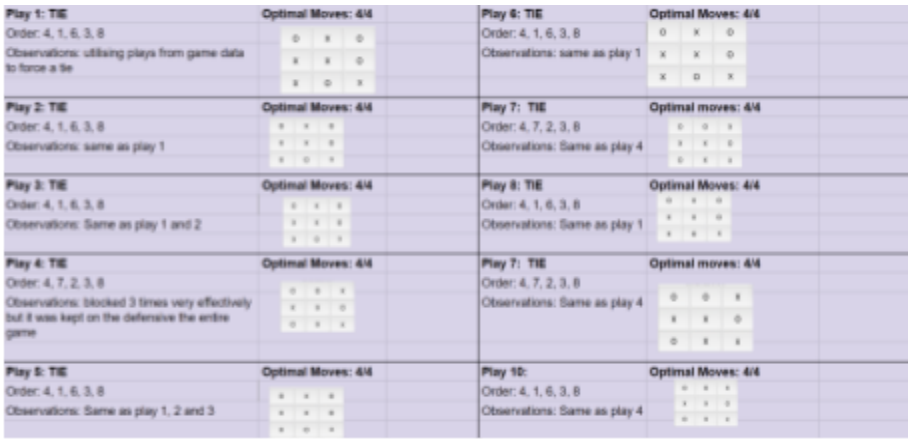


*Figure 14* - Boards are color-coded by similarity

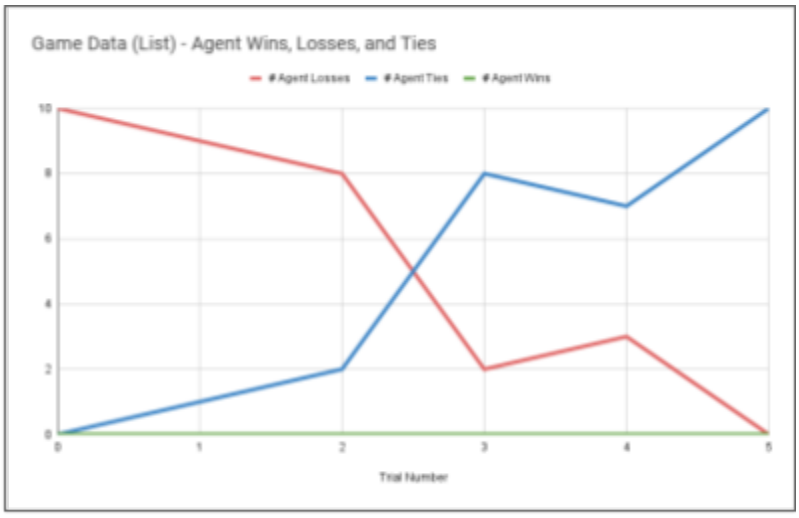


*Figure 15*

Including listed game data in the Agent prompt for each trial correlates with improved and even optimized gameplay. Providing information on previous patterns may have allowed the Agent to better assess board states and develop strategies for optimal play; thus, orders of moves that produced a tie could be repeated. The simple, listed structure of the game data may have also aided NLP efficiency in comprehending the prompts. Overall, this method correlates positively with Agent optimization.

## METHOD 3

Explaining game data from the previous trial in the Agent instructions produced minimal improvements in Agent gameplay. The *Probability of Agent Tie* increased within the first two trials, but reached a plateau and decreased slightly across the latter three. The *Probability of Optimal Move* increased in the second trial and immediately decreased generally showing no consistent pattern of increase or decrease throughout the other trials (Figure 16).

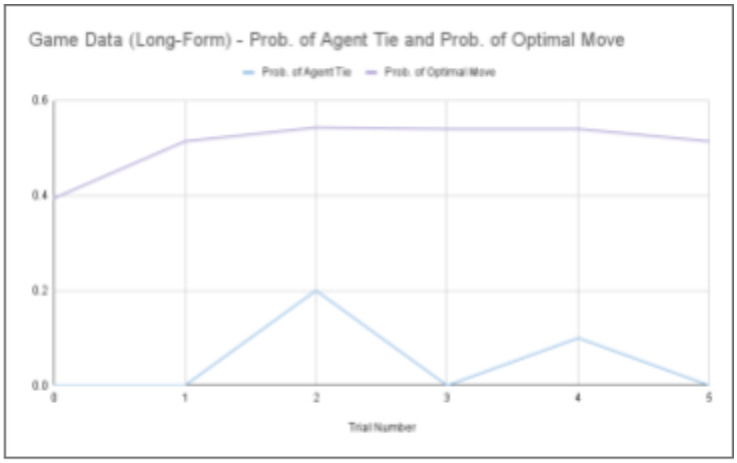


*Figure 16*

The Agent and the Optimal Agent played many of the same series of moves, resulting in dozens of repeated game boards. Certain game boards with more optimal Agent moves became more common as the trial number increased (Figure 17). However, the number of Agent Wins, Agent Losses, and Agent Ties did not clearly increase or decrease across the five trials (Figure 18).


*Figure 17 - Boards are color-coded by similarity*


*Figure 18*

Including written game data for each trial did not correlate to improved gameplay. One potential reason behind the discrepancy between this method and the previous could lie within the format of the game data; data explained in sentences could have contained added complexity, widening the margin of error for NLP interpretation. Thus, the Agent may not have comprehended the bulk of the game data because it was relayed inefficiently. Overall, the use of this method did not correlate with Agent optimization.

## METHOD 4

Listing game data from the previous trial in the Agent instructions (Method 2) as the Agent played against a Near-Optimal Agent produced strong improvements in gameplay, leading to optimal gameplay in Trial 5. The *Probability of Agent Win* increased and reached 1 in Trial 5. The *Probability of Optimal Move* increased slightly across all five trials. The *Probability of Agent Tie* increased, then decreased and reached 0. (Figure 19)

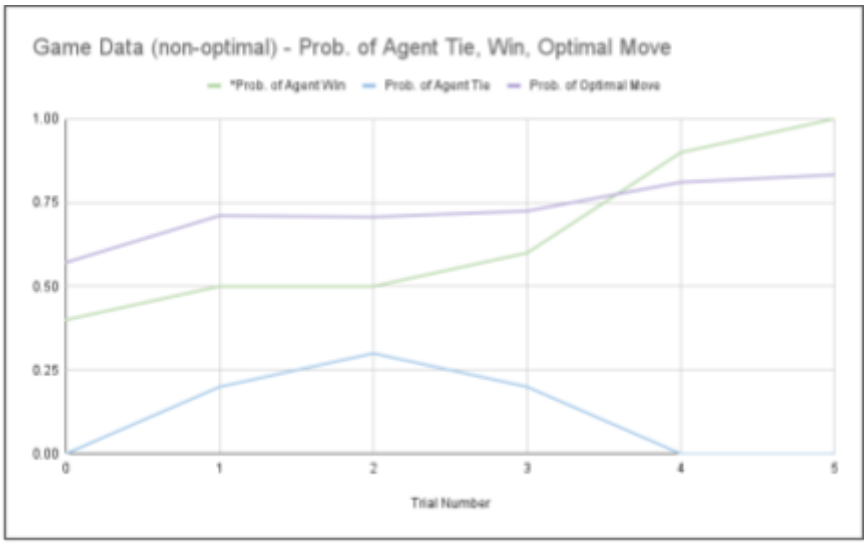


*Figure 19*

The Agent and the Near-Optimal Agent played several repeated series of moves; however, the game boards varied greatly (Figure 20). The number of Agent Wins increased and reached 1 while the number of Agent Losses and Agent Ties decreased and reached 0. (Figure 21). By Trial 5, the Agent played 100% optimally.

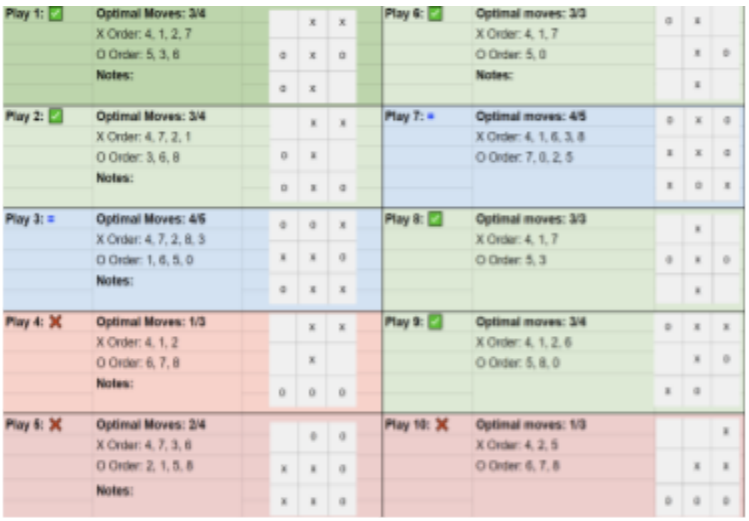


*Figure 20* - Boards are color-coded by similarity

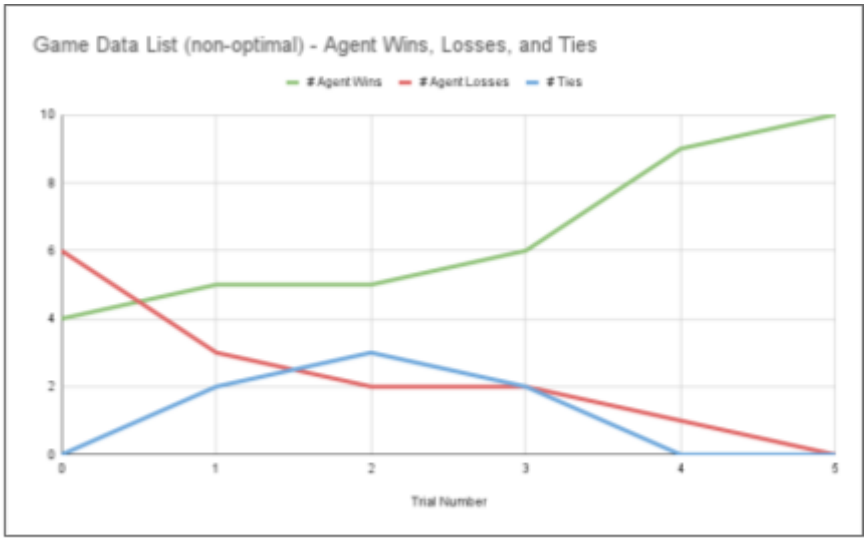


*Figure 21*

As with Method 2, including listed game data in the Agent prompt for each trial correlates with improved and even optimized gameplay. Playing the Agent against a Near-Optimal Agent resulted in vastly different game boards, repeated patterns, and possibilities of optimal moves, providing a diversity of data that could have greatly assisted Agent learning. Initially, both agents appeared evenly matched, but with each trial, the tested Agent gained a clear advantage. Overall, this method correlates positively with Agent optimization.

# Conclusions

In summary, Method 2 and Method 4 had the strongest correlation with optimized gameplay. Both methods utilized simple game data lists with short analyses of moves from each previous trial, allowing the Agent to "learn" from previous patterns of play. Method 1 and Method 3 did not clearly correlate with optimized gameplay. Both methods utilized sentence-based prompting that was relatively complex compared to the two successful methods. Thus, listed data within clear structures (refer to Method 2: Past Game Data List Form) are linked to the greatest improvement in Agent gameplay out of the four methods.

Ultimately, the findings demonstrate the effectiveness of careful, strategic, but simple prompt engineering in significantly enhancing LLM game logic performance. The initial untrained LLM played mostly nonoptimal moves and appeared to lack strategy. However, once the LLM was prompted specifically with past game data, its reasoning skills drastically improved, resulting in increasingly optimized gameplay. The straightforward nature of our research allowed us to swiftly observe gameplay data and trends. Our findings open up new applications for LLM prompt engineering, including complex game calculations, educational tools for logic and critical thinking, assistive human-like agents, and simulation training.

# Future Research

Future research will generalize our findings to more complex tasks and applications, including board games like Chess and Monopoly. Integrating emotion and sentiment analysis would enable LLMs to adjust their game decision-making based on emotional context, assuming human-like gameplay. In a broader sense, equipping LLMs with critical thinking skills and mistake-learning could prepare them to provide practical assistance to humans, specifically for elderly individuals or individuals with disabilities.

Artificial intelligence presents an exciting future. However, to improve our lives, AI must begin by improving itself.

# References


Google. “Prompt Design Strategies.” *Google AI for Developers*, 1 Aug. 2024,

ai.google.dev/gemini-api/docs/prompting-strategies.

Nicholls, Leon. “Tic-Tac-Toe and the Art of Gemini Prompt Engineering.” *Medium*, Medium, 7

Mar. 2024,

leonnicholls.medium.com/tic-tac-toe-and-the-art-of-gemini-prompt-engineering-0b0dfa4

7e733. Accessed 10 Aug. 2024.

RealPython. “Build a Tic-Tac-Toe Game with Python and Tkinter – Real Python.” *Realpython.com*, 2022, realpython.com/tic-tac-toe-python/.

TensorFlow. “Adversarial Example Using FGSM | TensorFlow Core.” *TensorFlow*, 19 July 2024, www.tensorflow.org/tutorials/generative/adversarial_fgsm.

Tong, Yongqi, et al. “Can LLMs Learn from Previous Mistakes? Investigating LLMs’ Errors to Boost for Reasoning.” *ArXiv (Cornell University)*, 29 Mar. 2024, https://doi.org/10.48550/arxiv.2403.20046. Accessed 4 Aug. 2024.

Topsakal, Oguzhan, et al. “Evaluating Large Language Models with Grid-Based Game Competitions: An Extensible LLM Benchmark and Leaderboard.” *arXiv*, 11 July 2024, arxiv.org/pdf/2407.07796.