ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

EVALUATING PERFORMANCE OF LARGE LANGUAGE MODELS IN BLUFF-BASED CARD GAMES: A COMPARATIVE STUDY

M.Sc. THESIS

İrem ŞALK

Department of Game and Interaction Technologies

Game and Interaction Technologies Programme

JUNE 2025

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

EVALUATING PERFORMANCE OF LARGE LANGUAGE MODELS IN BLUFF-BASED CARD GAMES: A COMPARATIVE STUDY

M.Sc. THESIS

İrem ŞALK
(529221007)

Department of Game and Interaction Technologies

Game and Interaction Technologies Programme

Thesis Advisor: Prof. Dr. Sanem Sarıel

JUNE 2025

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

BLÖF TEMELLİ KART OYUNLARINDA BÜYÜK DİL MODELİNE AİT PERFORMANS DEĞERLENDİRİLMESİ: KARŞILAŞTIRMALI BİR ÇALIŞMA

YÜKSEK LİSANS TEZİ

İrem ŞALK
(529221007)

Oyun ve Etkileşim Teknolojileri Anabilim Dalı

Oyun ve Etkileşim Teknolojileri Programı

Tez Danışmanı: Prof. Dr. Sanem Sarıel

HAZİRAN 2025

İrem ŞALK, a M.Sc. student of ITU Graduate School student ID 529221007 successfully defended the thesis entitled "EVALUATING PERFORMANCE OF LARGE LANGUAGE MODELS IN BLUFF-BASED CARD GAMES: A COMPARATIVE STUDY", which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Sanem Sarıel** Istanbul Technical University	.....
Jury Members :	**Assoc. Prof. Dr. Gökhan İnce** İstanbul Technical University	.....
	Assist. Prof. Dr. Emirhan Coşkun MEF University	.....

Date of Submission : **31 May 2025**
Date of Defense : **27 June 2025**

To my family and friends,

FOREWORD

This thesis was prepared as part of the requirements of the Master's program in Game and Interaction Technologies. I would like to thank my supervisor, Prof. Dr. Sanem Sarıel, for her valuable support throughout this research. I would like to also thank Uğur Önal for his support and professional guidance during this research.

I am sincerely grateful to my family, friends, and my cat for their constant encouragement and presence during this study. I also extend my heartfelt gratitude to my parents, who have taught me the value of science and academic work since my early childhood, and have provided unwavering financial and emotional support throughout the past 20 years of my education. Special thanks to Emre Mert for helping me stay motivated and for his invaluable mental support.

June 2025 İrem ŞALK

TABLE OF CONTENTS

ABBREVIATIONS

AI	**:** Artificial Intelligence
API	**:** Application Programming Interface
DQN	**:** Deep Q-Network
LLM	**:** Large Language Model
ICL	**:** In-Context Learning
MCTS	**:** Monte Carlo Tree Search
RL	**:** Reinforcement Learning
JSON	**:** JavaScript Object Notation

SYMBOLS

P_k	:	kth player
P'	:	Ordered set of players (based on dealer's right)
O_k	:	Set of opponents of P_k
t	:	Turn index
h_k^t	:	Hand of P_k at turn t
$c_{k,\text{mt.}}^t$	:	Matched cards in P_k's hand at t
$c_{k,\text{unm.}}^t$	:	Unmatched cards in P_k's hand at t
s^t	:	Game state at turn t
$\mathscr{T}$	:	Function to determine current player based on turn
a_c^t	:	Challenge action at turn t
a_m^t	:	Move action at turn t
δ_k^c	:	Challenge strategy function of P_k
δ_k^m	:	Move strategy function of P_k
e^t	:	Selected declaration vector at t
e_{default}^t	:	Default declaration when move not allowed
μ_k^t	:	Final move decision of player k at t
Y^t	:	Card pile at turn t
Y_k^t	:	Relative pile of cards played by player k
$Y_{k,\text{mt.}}^t$	:	Matched cards in Y_k^t
F^t	:	Frequency distribution of suits in move m^t
R^t	:	Outcome of challenge at t
$V(m^t)$	:	Outcome of bluff move
ω	:	Function returning the winner
w_k^t	:	Winner/loser status of player k at turn t
τ^t	:	Turn record at t

LIST OF TABLES

LIST OF FIGURES

Page

EVALUATING PERFORMANCE OF LARGE LANGUAGE MODELS IN BLUFF-BASED CARD GAMES: A COMPARATIVE STUDY

SUMMARY

The aim of this study is to investigate and compare the decision-making performance of multiple agent strategies in a modified bluff-based card game under imperfect information. The study is focused on whether a large language model (LLM) that is prompted through in-context learning (ICL), can generate effective action recommendations when comparisons are made to other agents that use traditional rule-based strategies and reinforcement learning in a game where deception plays a critical role. The central hypothesis is that successful bluff or challenge actions can be adaptively recommended by reasoning-driven agents using LLM by interpreting structured sets of information despite having no training on game-specific reward signals. Our research question is "Can LLM-driven strategy suggest optimal actions by making predictions and inferences about opponents in a bluff-based card game?"

This study is expected to contribute to growing research on the application of LLMs in real-time decision-making domains by analyzing the performance of the agents in terms of the success rate of actions and wins. To do that, the model's ability to reason over game states, player histories, and probabilistic cues to select actions in a bluff-based setting is focused on in the evaluation.

The game environment is a modified version of the traditional Bluff (also known as Cheat or BS) card game. The modified version of the Bluff game uses a special 24-card deck which consists of 6 suits and 4 cards and is played by three players (agents). At the start of the game, cards are evenly distributed among all players, the first player is chosen randomly and turns continue in a clockwise direction in each game. Cards are kept hidden by players from others during the game. A fixed table rank is used throughout the game. During the gameplay, two actions can be performed by players: challenge and move. When a move is made by players, cards are piled face down in the middle and a certain amount of cards with the table rank and rank value is claimed to be played. This feature allows opponents to be bluffed by players. A bluff move is defined as one in which at least one card does not match the declared rank, while a truthful move is one in which all cards match the declared rank. To simplify the action selection, the challenge phase is limited to one player, as the challenge action may only be performed by the next-moving player. Therefore, the previous move may be challenged by a player if it is believed that it does not match the required rank. If the last played move is a bluff, then all cards in the pile are taken by the previous player; otherwise, all cards in the pile are taken by the player who performs the challenge. The first player who discards all the cards in their hand is defined as the winner of the game. The game is proceeded in turns until one player wins.

To create a game simulation framework, the gameplay and rules are first modeled mathematically by using set notation based on our game design. Then, the frequency of action selections along with their corresponding outcomes is derived, and dynamic reward-penalty for each action and game conclusion is defined. Next, the game is modeled which is played by five different agents. Actions are selected by each agent based on its internal strategy logic, and rewards are distributed based on the success of bluffs and challenges. In this methodology, five distinct agents are implemented:

1. Random Agent: This agent selects actions uniformly at random without considering game context. Serves as a performance baseline.

2. State Dependent Agent: This agent uses handcrafted rules based on the current state, such as pile size, card distribution. The agent with minimal logic and contextual awareness as calculating the probability of occurrence of the matching cards in the opponent's hand is modeled.

3. Bayesian Agent: Employs hierarchical priors from historical game data and current game state to evaluate best action for bluffing and challenging. Adapts action preferences based on prior success rates.

4. DQN Agent: A deep reinforcement learning model trained to maximize long-term reward. It maps observed states to derive optimal actions using Q-learning, updating its policy across turns in episodes. To support reinforcement learning, we defined both episode-level rewards (e.g., winning or losing the game) and turn-based rewards for each action type (move and challenge) to guide the learning process of the agent effectively. We design two different learning configurations as Baseline-Oriented Training which is competing against baseline agents to learn stable behavior in known scenarios and self-play training which provides learning by playing against versions of itself to promote generalization and adaptability.

5. LLM Agent: A GPT-based language model (GPT-4o) receives a structured prompt describing the rules and instructions, current state, past behaviors, and probabilistic game summaries such as winning rate. In addition, a simple chain of thoughts by instructions and strategy guidelines to analyze opponent behaviors, predict possible outcomes of possible action scenarios and evaluate risk-reward to select an action is design and implemented. It reasons step-by-step internally but outputs only the final action.

To ensure fair evaluation of the LLM agent, an opponent filtering mechanism is implemented, where the LLM is only tested against a selected pool of agents with varied but stable behaviors. This prevents high variance due to opponent unpredictability and allows consistent measurement of reasoning-based performance.

The conducted simulation results are analyzed by strategy type, focusing on bluff/challenge success, consistency, and win rate. Action success rates and cumulative performance are included in the metrics. The analysis is conducted by comparing static (Random, State-dependent, Bayesian), learning-based (DQN), and reasoning-based (LLM) agents to determine how performance in uncertain, deceptive environments is influenced by adaptive decision-making.

By comparing action decisions of the LLM agent with those of the other strategies, the effectiveness of LLM-driven recommendations is evaluated in the study. The simulation framework is contributed to for understanding how LLMs perform in strategic decision-making tasks where uncertainty and deception are key components.

BLÖF TEMELLİ KART OYUNLARINDA BÜYÜK DİL MODELİNE AİT PERFORMANS DEĞERLENDİRİLMESİ: KARŞILAŞTIRMALI BİR ÇALIŞMA

ÖZET

Bu tez çalışmasının amacı, kısmi olarak gözlemlenebilen (partially observable) bir oyun ortamında çeşitli etmen stratejilerinin karar verme performanslarının incelenmesi ve karşılaştırılmasıdır. Özellikle, blöf unsurunun kritik rol oynadığı bir oyunda, bağlam içi öğrenme (in-context learning, ICL) yoluyla yönlendirilen büyük dil modellerinin (LLM) geleneksel kural tabanlı stratejiler ve pekiştirmeli öğrenme ile eğitilen etmenlere kıyasla etkili eylem önerileri üretip üretemediği araştırılmıştır. Bu kapsamda, temel hipotez olarak, oyunla ilgili özel bir ödül sinyali ile eğitilmemiş olmasına rağmen, yapılandırılmış bilgi kümelerinin yorumlanması yoluyla dil modeli tabanlı etmenlerin başarılı blöf veya itiraz eylemleri önerebildiği varsayılmıştır. Bu bağlamda, tezin araştırma sorusu "Blöf temelli bir kart oyununda, Büyük Dil Modeli bazlı bir stratejinin, rakipleri hakkında çıkarım yaparak en iyi eylemleri önerebilirliği" şeklinde ifade edilmiştir.

Bu çalışma kapsamında, büyük dil modellerinin gerçek zamanlı karar verme alanlarındaki uygulanabilirliği analiz edilerek literatüre katkıda bulunulması hedeflenmiştir. Bu amaçla, etmenlerin blöf gerektiren eylemleri sergileme sıklıkları, diğer oyuncuların blöflerini tahmin edebilme sıklıkları, bu eylemlerin başarılı olma oranları ve oyun kazanma oranları üzerinden performansları değerlendirilmiştir. Değerlendirme aşamasında, modelin oyun durumları, oyuncu geçmişleri ve olasılıksal ipuçları üzerinden akıl yürüterek eylem seçme yeteneğine odaklanılmıştır.

Geliştirilen oyun ortamı, geleneksel Blöf oyununun değiştirilmiş bir versiyonu olarak tasarlanmıştır. Bu versiyonda, altı farklı türde dörtlü kartlardan oluşan 24 kartlık özel bir deste kullanılmış ve üç oyuncu (etmen) tarafından oynanması sağlanmıştır. Oyun başında kartların oyuncular arasında eşit olarak dağıtılması ve ilk oyuncunun rastgele belirlenmesi sağlanmıştır; tur sırasının saat yönünde ilerlemesi kural olarak benimsenmiştir. Oyuncuların ellerindeki kartların diğer oyunculardan gizli tutulması sağlanmıştır. Oyun boyunca sabit bir masa rengi kullanılmış ve bu renk, kartların sahip olduğu renkler arasından rastgele seçilmiştir. Her turda oyuncular tarafından gerçekleştirilebilen iki eylem tanımlanmıştır: kart oynama (hamle) ve rakibin hamlesine itiraz etme. Hamle yapıldığında, kartların yüzü kapalı olarak ortaya bırakılması ve oynanan kartların masa rengi ile aynı olduğu iddia edilmesi zorunlu kılınmıştır. Bu iki hamle kuralı sayesinde, oyuncuların rakiplerine blöf yapmasına ve iddia ettikleri renkten farklı kartlar oynamasına olanak tanınmıştır. Bir blöf hamlesi, en az bir kartın ilan edilen kart rengi ile eşleşmemesi durumunda gerçekleştirilmiş sayılmıştır; gerçek bir hamlede ise tüm kartların bu masa rengi ile eşleşmesi gerekmektedir. Oyuncuların ellerinde masa rengi ile uyuşan kart bulunmasa dahi hamle yapmak zorunda olmaları, oyuncuların kazanmak için doğru şekilde blöf yapmaya itilmesi şeklinde kurgulanmıştır. İtiraz

eylemi opsiyonel olarak tanımlanmış ve oyuncu rakibin blöf yaptığını düşündüğünde hamlesine itiraz edebilmiştir. İtiraz eylemi gerçekleştirildiğinde, son oynanan kartların içeriğine bakılmıştır. Eğer rakibin blöf yaptığı tespit edilirse, ortadaki kartlar ceza olarak rakip oyuncuya verilmiştir. Aksi halde, itiraz eden oyuncuya ceza verilmiştir. Oyun, bir oyuncunun elinde kart kalmadığında ve bu oyuncu rakibin itirazı ile ceza almadığında sonlandırılmıştır. Oyuncunun elindeki tüm kartları elinden çıkararak oyunu kazanması, diğer oyuncuların ise kaybetmesi ile sonuçlanmıştır.

Bu tezde, oyun benzetim çerçevesi oluşturulabilmesi amacıyla öncelikle oyun kuralları ve oynanışı küme gösterimi ile matematiksel olarak modellenmiştir. Ardından, eylem seçimlerinin frekansları ve sonuçları türetilmiştir. Bunun için, her oynanan oyun için tur bazlı oyuncu eylem seçimlerini tutan ve bunları istatistiksel olarak hesaplayan bir model tasarlanmıştır. Benzetim kapsamında beş farklı etmen tipi modellenmiştir. Her bir etmenin strateji mantığına göre eylem kararı aldığı varsayılmıştır. Tanımlanan etmen türleri şunlardır:

1. Rastgele Etmen, oyun bağlamını dikkate almadan eylemleri rastgele seçen bir temel referans olarak kullanılmıştır.

2. Duruma Bağlı Etmen, o anki oyun durumuna (örneğin ortaya koyulan kart sayısı, kart dağılımı) göre rakiplerin ellerinde eşleşen kartların olasılıklarını değerlendirerek basit bağlamsal mantık uygulamıştır.

3. Bayesçi Etmen, eylem kararı alırken geçmiş oyun verilerinden elde edilen hiyerarşik öncülleri ve mevcut oyun durumunu kullanmıştır.

4. Pekiştirmeli Öğrenme Etmeni (Derin Q-Ağı etmeni, DQN), Q-öğrenme algoritması kullanılarak uzun vadeli ödülleri en üst düzeye çıkarmaya çalışan derin öğrenme tabanlı bir model olarak tasarlanmıştır.

5. Büyük Dil Modeli Bazlı Etmen ise, yapılandırılmış bilgi girdisi aracılığıyla kararlar üreten, kuralları, mevcut durumu, geçmiş davranışları ve olasılıksal özetleri kullanarak stratejik akıl yürütme yapan bir model olarak geliştirilmiştir.

Derin Q-Ağı etmeni için iki farklı öğrenme yapılandırması tanımlanmıştır: baz stratejileri kullanan etmenlerle öğretilme yöntemi ve kendi kendine oynama yöntemi. Ek olarak, hem hamle hem de itiraz eylemleri için tur bazlı ve bölüm bazlı ödül tanımlamaları yapılmıştır. Böylece, Derin Q-Ağı etmeninin performansı farklı öğrenme senaryolarına göre değerlendirilmiş ve iyi performans gösteren etmenler diğer etmenlerle karşılaştırılmak üzere seçilmiştir.

Dil modeli bazlı eylem seçimi yapan etmenin adil şekilde değerlendirilebilmesi için, yalnızca davranışları optimal olan ve oyunlar arasında başarılı sonuçlar elde eden etmenlerin seçildiği bir rakip havuzuna karşı test edilmesi sağlanmıştır. Böylelikle, yalnızca iyi performans gösteren etmenler kullanılarak yüksek varyans kaynaklı dengesizliğin önlenmesi ve akıl yürütme bazlı kararların daha sağlıklı ölçülmesi mümkün kılınmıştır. Bu filtrelemeden elde edilen en iyi performansa sahip etmenler üç gruba ayrılarak dil modeli etmeni ile oyun ortamında benzetimler gerçekleştirilmiştir. Bu oyuncu gruplamaları sırasıyla, yalnızca Derin Q-Ağı etmenleri ile dil modeli

etmeni, yalnızca tanımlanan baz stratejileri kullanan etmenler ile dil modeli etmeni ve her iki benzetimde de en iyi performans gösteren etmenler ile dil modeli etmeni şeklinde tanımlanmıştır. Böylelikle, farklı oyun tarzlarında dil modeli tabanlı etmenin performansı analiz edilmiştir.

Dil modeli etmeni olarak GPT-4o tabanlı model kullanılmıştır. Bu modele yapılandırılmış bir girdi aracılığıyla kurallar, mevcut durum, geçmiş davranışlar ve olasılıksal özetler sunulmuştur. Ayrıca, rakip davranışlarının analiz edilmesi, olası sonuçların tahmin edilmesi ve risk-ödül dengesinin değerlendirilmesi amacıyla stratejik yönergeler ve düşünce zinciri temsili sağlanarak adım adım akıl yürütme yapılması ve nihai kararın verilmesi sağlanmıştır.

Benzetim sonuçları strateji türüne göre analiz edilmiştir. Analizler, blöf ve itiraz başarı oranları, tekrar eden denemelerde tutarlılık ve genel kazanma oranlarına odaklanır. Rastgele, kural tabanlı, öğrenme temelli ve akıl yürütme temelli etmenler karşılaştırılarak belirsizlik ve blöfün hakim olduğu ortamlarda, karar verme yetilerinin etkisi değerlendirilmiştir.

Dil modeli etmeninin kararlarının diğer stratejilerle karşılaştırılması yoluyla, yapılandırılmış yönlendirme aracılığıyla dil modeli temelli stratejilerin etkinliği değerlendirilmiştir. Oluşturulan bu benzetim çerçevesi, büyük dil modellerinin stratejik karar verme ortamlarında nasıl performans gösterdiğine ve blöf içeren durumlara nasıl yanıt verdiğine dair bir analiz sunmaktadır. Bu çalışma, dil modeli ile eylem seçen etmenin eylem başarıları ve kazanma sıklığının yanı sıra, bu eylemleri seçme sıklığını da değerlendirmiştir. Bu sayede, yanıltıcı bir ortamda, oyuna dair verilen güncel ve geçmiş bilgileri analiz eden dil modelinin nasıl bir oyuncu davranışı sergilediği de gösterilmiştir. Sonuç olarak, modellediğimiz benzetim çerçevesi, belirsizlik ve blöfün temel bileşenler olduğu stratejik karar alma görevlerinde büyük dil modellerinin nasıl performans gösterdiğini anlamaya katkıda bulunur. Sonuçlar, dil modeli etmenlerinin, yeterli yönlendirme sağlandığında, en iyi eylemlere yakın sonuçlar üretebildiğini ve gelecek çalışmalarda oyun planlama, yönlendirme yapısı ve belirsizliğin modellenmesi gibi alanlarda daha ileri analizlerle bu performansın derinleştirilebileceğini göstermektedir.

1. INTRODUCTION

This chapter introduces the research motivation and outlines the primary goals of this thesis. It begins by explaining the purpose and the scope of the study, focusing on the evaluation of decision-making agents in bluff-based card games under imperfect information. The following sections describe the contributions made to agent modeling, simulation framework design, and the integration of large language models (LLMs) into strategic gameplay. The chapter also emphasizes the importance of comparing heterogeneous agents under a unified environment. These elements together form the foundation for exploring agent behavior in deceptive multi-agent environments.

1.1 Purpose of Thesis

The primary purpose of this thesis is to investigate how LLM agents perform against different agent architectures in bluff-based card game environments under imperfect information. To do this, this research aims to compare traditional probabilistic, learning-based, and reasoning-driven agents to evaluate their effectiveness in strategic decision-making scenarios that require both deception and inference. In this way, the agent using large language model (LLM) is evaluated for its potential to make contextually informed decisions through in-context learning (ICL), without relying on explicit training with game-specific reward signals.

Strategic decision-making in games under imperfect information presents a challenging yet insightful domain for evaluating the capabilities of artificial agents. Bluff-based card games, in particular, require agents to make calculated decisions based on partial observations, probabilistic reasoning, and predictions about opponents' hidden states and intentions. In this context, bluff-based card games offer a controlled yet complex environment where agents must evaluate partial information, choose or detect between truthful or deceptive actions based on their inference. This complexity gives an opportunity to test the LLMs capability without fine tuning or training, particularly in

how they interpret structured data and generate behavior when faced with deceptiveness by others.

The thesis proposes a simulation framework where agents make sequential decisions, either making a move or challenging an opponent's move, with incomplete knowledge about other players' hands. The environment has been mathematically defined and algorithmically implemented to support fair comparisons. Five distinct agent types are considered: Random Agent, State-Dependent Agent, Bayesian Agent, DQN-based Reinforcement Learning Agent, and LLM Agent using GPT-4o with structured prompts.

A secondary but equally important goal of the thesis is to explore whether natural language-based agents can achieve comparable decision quality to reinforcement learning agents. This includes analyzing whether structured prompt design, combined with probabilistic summaries and behavioral history, can guide the LLM agent to reason effectively in game-like scenarios. Additionally, the interpretability of LLM agents is considered a potential advantage in use cases where the transparency of reasoning is critical, such as human-agent interaction or educational game applications.

To sum up, the purpose of this thesis is twofold: first, to establish a unified simulation environment to evaluate heterogeneous decision-making agents in a deception-centered strategic setting; and secondly, to assess whether LLMs can perform competitively against optimized agents without direct reward-based training, only based on reasoning and structured prompts. Through this analysis, the study aims to contribute to ongoing discussions in AI research on explainable, adaptive, and generalizable decision-making in strategic, partially observable environments.

1.2 Contributions

This thesis provides both conceptual and technical contributions to the study of decision-making agents in bluff-based card games under imperfect information. The key contributions of this research are as follows:

1. **Formalization of a Modified Bluff-Based Card Game Environment:** A mathematically grounded simulation environment is designed, incorporating key bluffing mechanics and hidden information.

2. **Unified Framework for Agent Comparison:** A consistent and extensible simulation platform is implemented to evaluate the performance of heterogeneous agents—including Random, Conditional, Bayesian, DQN-based reinforcement learning, and in context learning-based LLM agents under same conditions. This ensures fair comparison across architectures and decision-making paradigms.

3. **Integration of Large Language Models (LLMs) into Strategic Gameplay:** The thesis introduces a novel use of GPT-4o as a decision-making agent using structured prompts without gradient-based training or reward feedback. The LLM agent's decisions are generated through in-context reasoning based on numeric, probabilistic and statistical summaries about agent and opponents behaviors.

4. **Structured Prompt Design for In-Context Reasoning:** A strategy-guided prompt template is developed to represent game state, historical context, strategy guidelines in a form interpretable by the LLM. This contribution explores the capabilities and limitations of LLMs in reasoning over partially observable multi-agent environments.

5. **Empirical Comparison of Agent Types:** Experiments are conducted to evaluate win rate, action accuracy, and deception resolution capabilities across all agent types. The results reveal how LLM agents perform compared to baseline or reinforcement learning agents, and under what conditions they succeed or fail.

6. **Interpretability and Behavioral Analysis of LLM Agents:** In addition to performance metrics, the reasoning traces (text outputs) of the LLM agent are analyzed to understand its decision pathways and error patterns. This analysis provides insights into the transparency and plausibility of language-based agents in strategic environments.

7. **Contribution to General AI Evaluation in Games:** The research contributes to the broader field of explainable and generalizable game-playing agents and proposing new methods to benchmark non-learning agents against trained models.

2. LITERATURE REVIEW

This chapter presents a review of existing research on strategic decision-making in games, with a focus on methods applied to bluff-based and imperfect information environments. In addition, it outlines the foundational methodologies behind Deep Q-Networks and Large Language Models, which are central to the agent strategies investigated in this study.

2.1 Related Works

Research on strategic decision-making in games has evolved through various methodological paradigms, including Monte Carlo Tree Search (MCTS), deep reinforcement learning (DRL), and most recently, large language models (LLMs). In this section, we group prior work based on the underlying method employed, with a particular focus on bluff-based or imperfect information games.

A study that modeled bluffing behavior in games through simulation-based methods is by Cowling et al. [1], where MCTS is applied to study inference and deceptive play. In their work, bluffing is treated as an emergent behavior resulting from the agent's probabilistic simulations over possible opponent actions and hidden states. This research indicates the potential of search-based methods to capture strategic elements such as inference and uncertainty in competitive environments.

Several studies have employed reinforcement learning techniques to address decision-making in partially observable games. Luo and Tan [2] introduce a DRL-based framework for mastering DouDiZhu, a multi-agent card game characterized by bluff and hidden information. Their architecture integrates self-play, multi-agent coordination, and deep Q-learning variants to achieve competitive-level performance. The study demonstrates how value-based reinforcement learning agents can be effective in strategic reasoning even in environments with incomplete information.

Recent advancements in large language models (LLMs) have opened up new opportunities for reasoning-based interactive gameplay and opponent modeling. LLM performance under in context learning works represents capacity of LLMs' to act as game-playing agents, either through prompt-based decision making. Huang et al. [3] present PokerGPT, an end-to-end LLM-based solver for multiplayer Texas Hold'em. The model is trained with both task-specific prompting and fine-tuning to balance bluffing, hand strength estimation, and opponent modeling in an end-to-end fashion. Guo et al. [4] propose the Suspicion-Agent in their study, which uses GPT-4 to simulate theory-of-mind reasoning in imperfect information settings. Their experiments demonstrate that LLMs can adapt to opponent belief structures and respond strategically to potential deception by providing only game contexts and opponents' statistics to the reasoning agents.

Duan et al. [5] introduce GTBench, a benchmark suite for evaluating LLMs on their ability to perform strategic reasoning in various game-theoretic scenarios. Their results show fundamental reasoning limitations of LLMs when faced with long-term dependencies and opponent adaptation.

Wu et al. [6] design an instruction-following poker engine powered by LLMs, showing how structured natural language inputs can drive game agents' decisions. This work emphasizes the potential for instruction-tuned LLMs to integrate seamlessly with game engines for adaptive play.

Hu et al. [7] provide a comprehensive survey on LLM-based game agents, categorizing current capabilities and identifying open challenges in perception, planning, and language-grounded interaction.

Also, LLM can be used as guidance for players, which Jataware [8] explores this by modeling LLM-driven assistant agents for the game of Diplomacy, where the model provides real-time suggestions based on evolving state and dialogue between players.

While MCTS and DRL methods offer interpretable and high-performing solutions in well-defined settings, LLMs provide flexibility and adaptability through in-context reasoning. However, recent evaluations such as GTBench [5] underscore that LLMs still

face challenges in maintaining consistent strategic coherence and reasoning, particularly in multi-agent and imperfect information scenarios.

2.2 Large Language Models

2.2.1 Large Language Models and transformer architecture

Large Language Models (LLMs) are a class of artificial intelligence systems trained on massive corpora of textual data using deep neural networks, primarily transformer architectures. These models are designed to understand and generate human-like language by predicting the next token in a sequence given its context. Modern LLMs, such as OpenAI's GPT (Generative Pre-trained Transformer) series, have demonstrated remarkable capabilities across a wide range of natural language tasks including question answering, summarization, code generation, and dialogue [9].

At the core of LLMs is the *Transformer architecture*, first introduced by Vaswani et al. [10]. This architecture relies heavily on a mechanism called *self-attention*, which enables the model to assign dynamic weights to each token in a sequence based on its relevance to others. This allows the model to capture long-range dependencies and contextual relationships more effectively than prior architectures such as RNNs or CNNs. Multi-head attention, layer normalization, and residual connections are key components that help scale transformers to billions of parameters while maintaining training stability and performance.

2.2.2 In-context learning (ICL)

A key paradigm that has emerged with the advancement of LLMs is *In-Context Learning* (ICL). Unlike traditional machine learning approaches that require parameter updates through fine-tuning on labeled datasets, ICL enables models to perform tasks by conditioning on a prompt that includes example inputs and outputs [9,11]. In this setting, the model is not updated; rather, it infers patterns and generalizes directly from the context it is provided at inference time.

The process of ICL operates through the following principles:

1. **Prompting**: The model is presented with a sequence of demonstration examples in the form of input-output pairs, followed by a new input for which an output is expected.

2. **Pattern Recognition**: The model uses its internal learned representations to infer relationships between inputs and outputs in the examples.

3. **Generalization**: The model generates a response for the new input based on the inferred patterns, effectively simulating task-specific learning without modifying its parameters.

Recent work by OpenAI [12] has outlined practical strategies for maximizing performance in ICL settings, emphasizing the role of structured prompting methods. One prominent approach is Chain-of-Thought (CoT) prompting, where the model is guided to reason step-by-step before producing a final answer [13,14]. This technique has been shown to significantly improve performance on complex decision-making and reasoning tasks, including those involving arithmetic, logic, and strategic planning.

Another technique that is called Strategy Guidance, involves incorporating explicit reasoning frameworks or conditional rules into the prompt to influence the model's decision pathway [12]. By embedding principles, evaluation heuristics, or hypothetical cases within the prompt, the model can be steered toward more interpretable and goal-aligned outputs. These methods are particularly effective in domains such as multi-agent interactions, games, or decision support systems, where understanding context-specific dynamics is critical.

OpenAI's GPT-4 architecture, particularly the GPT-4 Omni (`gpt-4o`) variant, represents one of the most capable publicly accessible LLMs as of 2025. GPT-4o integrates improvements in inference speed, cost-efficiency, and multimodal capabilities. It can reason over both structured and unstructured data and generate consistent, instruction-aligned outputs across diverse use cases. GPT-4o, like its predecessors, supports ICL and exhibits strong few-shot and zero-shot generalization performance [12].

Despite their impressive capabilities, LLMs also have limitations. One of the key issues is *hallucination*, where the model generates factually incorrect or logically invalid outputs with high confidence [15]. Additionally, *context window limitations* restrict the amount of information the model can process at once, which can hinder performance in tasks requiring long-term memory or multi-step reasoning [9]. Furthermore, LLMs do not possess true understanding or beliefs; they operate purely based on learned statistical patterns, which can result in inconsistent behavior or failures in edge cases [15].

2.3 Deep Q-Network (DQN) as Reinforcement Learning Agent

Reinforcement Learning (RL) is a learning paradigm in which agents interact with an environment by taking actions and receiving feedback in the form of rewards. The objective of an RL agent is to learn a policy that maximizes cumulative rewards over time. Among various RL algorithms, the Deep Q-Network (DQN) [16] has emerged as a foundational method that integrates Q-learning with deep neural networks.

In DQN, the agent approximates the action-value function $Q(s,a)$ using a deep neural network, where s denotes the current state and a is the possible action. At each step, the agent selects actions based on an ε-greedy policy, balancing exploration and exploitation. After executing an action, the agent observes the reward and the next state, and stores this transition in a replay buffer. The network is then updated using mini-batches of experiences sampled from this buffer to reduce correlation and improve stability.

A target network is used alongside the primary Q-network to compute stable target values, which helps mitigate issues such as overestimation bias. The network is trained by minimizing the temporal-difference (TD) loss between predicted Q-values and target Q-values, based on the Bellman equation:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathscr{D}}\left[\left(r + \gamma \max_{a'} Q_{\text{target}}(s',a') - Q_\theta(s,a)\right)^2\right] \tag{2.1}$$

DQN has demonstrated significant success in a variety of domains, most notably in learning to play Atari 2600 games directly from pixels [16]. Since its introduction, several improvements and variants have been proposed to address its limitations. These include Double DQN [17] to reduce overestimation, Dueling DQN [18] to separate

value and advantage estimation, and Prioritized Experience Replay [19] to sample more informative transitions.

These extensions improve learning efficiency and policy stability, making DQN and its variants suitable for tasks involving discrete action spaces and complex state representations.

Despite its success, DQN also has known limitations. It is less effective in environments with large or continuous action spaces, where methods like policy gradient or actor-critic approaches may be more suitable. DQN also requires careful tuning of hyperparameters, and its performance can be sensitive to the design of the reward function, exploration strategy, and network architecture [20].

Additionally, DQN agents tend to struggle in non-stationary or multi-agent environments, where the assumptions of stationarity and single-agent optimization break down. These challenges motivate the exploration of more adaptive, robust learning algorithms for strategic settings such as bluff-based games.

3. GAME ENVIRONMENT

This chapter formally defines the structure and decision dynamics of a bluff-based card game environment under conditions of imperfect information.

3.1 Problem Description

The central research problem involves modeling a turn-based multiplayer game where players are required to either deceive their opponents or detect deception to succeed. Unlike deterministic or fully observable games, bluff-based games involve chance-based initial conditions, hidden actions and partial observability, which make the strategic evaluation and learning more complex.

To address this, we design a custom version of the Bluff card game with constrained rules that simplify the action space while preserving the core mechanic of deception. The chapter introduces the game environment, including rules, flow, and key design decisions (e.g., fixed table color, no-pass turns), followed by a formal mathematical representation of all core elements such as players, cards, moves, challenges, hands, and the evolving game state. This formalization enables the implementation of agents that can perceive, act, and learn in the environment through decision-making functions. The goal is to construct a controlled yet rich environment that allows systematic evaluation of agent strategies in bluff-based settings.

3.2 Traditional Bluff Card Game

Traditional Bluff game, which is also known as Cheat, I Doubt It, and BS, is a card game that is played by three to six players with one or two decks of cards. The primary goal of the game is to be the first player who empties their hand. To get rid of all the cards in a hand, each player plays a set of cards face-down and must announce the ranks of the cards they play with a twist of bluffing in order to deceive the other players.

Although the core objective of the Bluff game is always to be the same, which is to be the first player to discard all cards, remains the same across various versions of the game [21], [22]. These differences primarily revolve around two design elements: the declared color on each turn and whether playing a move is mandatory. In some variants, players must follow a strict ascending or descending sequence when declaring ranks, while others allow the player who became successful after a challenge to have the freedom to select rank or reselect the table rank if everyone in the game passes to make a move. Additionally, certain versions require players to make a move on every turn, effectively forcing a bluff when no valid cards are available. In contrast, other variants allow players to pass strategically.

Beyond these structural rule differences, the core gameplay loop and psychological tension in Bluff rely on player interaction and the balance between deception and deduction. Each round consists of a play phase and a challenge phase. During the play phase, a player must choose one or more cards to place face-down in the discard pile and declare a rank aloud. They may either tell the truth or lie about the cards played. In the challenge phase, any of the other players may call a "bluff" if they believe the cards played do not match the declared rank. If a challenge is issued, the face-down cards are revealed:

- If the declaration was false, the bluffing player must pick up the entire discard pile.
- If the declaration was true, the challenger must pick up the entire discard pile.

The player who wins the challenge typically begins the next round, or in some variants, gains the authority to set the next valid rank for play.

To create more dynamic player strategies, some implementations of the game allow for special rules:

- Flexible Rank Sequences: Players may select any rank, or only adjacent ones (e.g., one above or below the last played).
- Pass Mechanics: When passing is allowed, players can feign weakness or manipulate turn order to pressure others into bluffing.

- Wildcard or Joker Rules: In some digital or house versions, jokers or special “wild cards” are added, allowing instant success or forced bluff counters.

A critical part of the design is hand management and deduction. As the discard pile grows, it becomes increasingly difficult to recall which cards have been played or are held by others. Skilled players track played cards, observe behavioral cues, and time their challenges for maximum advantage. Meanwhile, novice players may fall into predictable bluffing or over-challenging patterns, providing a natural skill progression over repeated plays.

Also, this game provides the player a different starting point each time, since the initial conditions of the game depend on probability. The random distribution of cards at the beginning of each match ensures that no two games are identical. These probabilistic starting conditions prevents memorization while ensure that no two games are exactly alike, introducing variability in both card availability and early-game strategy. As a result, players are encouraged to think flexibly and adapt their tactics according to the specific boundary conditions of each match. Moreover, the element of chance is not limited to the initial setup; it also plays a crucial role throughout the gameplay by influencing how players interpret and respond to others’ actions. Since all actions are made under conditions of uncertainty such as where players may choose to bluff or tell the truth and each decision is inherently probabilistic since they also need to think about opponent’s action as a response to themselves. This dynamic requires players to exercise inference skills to deduce opponents’ intentions and behavioral patterns over time. Thus, we can state that a successful gameplay depends not only on managing one’s own cards but also on accurately interpreting the likelihood of deception and challenging in others and pile on the middle. This features provides a continuous process of probabilistic reasoning and strategic adaptation to the game. This randomness enhances re-playability and strategic depth by requiring players to continuously reassess risk, bluffing potential, and opponent modeling based on incomplete and evolving information.

3.3 Custom Bluff Game for Simulation Environment

A custom simulation environment based on a modified version of the Bluff card game is used, and it is structured to model imperfect information and deception dynamics.

Several game elements and rules are modified to reduce the complexity of the action space and the session length, standardize the flow of turns, and enable controlled evaluation of deception and inference under imperfect information. Thus, the custom version designed for this simulation departs from the classical Bluff game in several key ways:

1. It uses a fixed suit color (table color) instead of rotating ranks,
2. Passing is not allowed—players must make a move every turn,
3. A reduced custom deck with four ranks and six suits is used to simplify the action space,
4. The gameplay flow enforces a strict challenge-then-move structure on every turn.
5. Players can challenge only the last move before their turn.

Game rules and action definitions are given as a list below:

1. Number of players: The game can be played with three players.
2. Card distribution: All cards are dealt to the players in a cyclical clockwise manner. If the cards cannot be distributed evenly, the difference in the number of cards held by any two players must not exceed one.
3. Selection of card dealer and first player: A dealer is chosen randomly. The first player is the one who receives the first card during the deal.
4. Hand showing: Players are not allowed to reveal their hands to others.
5. Turn order: Players take turns in a clockwise direction.

6. Table color: The table color is determined before the game begins and it is selected randomly between all suit colors and remains fixed throughout the game. This is required color to be play in each turn.

7. Move announcement: A player must declare the table color and the number of cards they are playing.

8. Playing a move: During their turn, a player must place one or more cards face-down on the pile and declare the table color.

   (a) Bluff move: A move includes at least one card not matching the declared color.

   (b) Truthful move: A move in which all cards match the declared color.

9. Challenge: A player may challenge the previous move if they believe it does not match the required color. Challenges cannot be made if there are no cards in the pile or a previous move has not been made.

   (a) Successful challenge: If the challenged move was a bluff, the challenger succeeds, and the previous player takes all the cards in the pile as a penalty. The challenger continues to play their turn.

   (b) Unsuccessful challenge: If the challenged move was truthful, the challenger fails and must take all the cards in the pile as a penalty. The challenger also forfeits their turn.

10. Turn objective: On their turn, players have two available actions: challenge and move. Players are not required to challenge their opponents, but they must move if they can (no failed challenge actions on turn).

11. Game objective: If a player plays their final card(s) and a subsequent challenge is initiated by the next player, the outcome of the challenge does not affect the game result if the player remains without cards after resolution. The game ends with that player as the winner. All remaining players lose.

As seen in Figure 3.1, the gameplay proceeds in repeated turns until one player wins by emptying their hand.

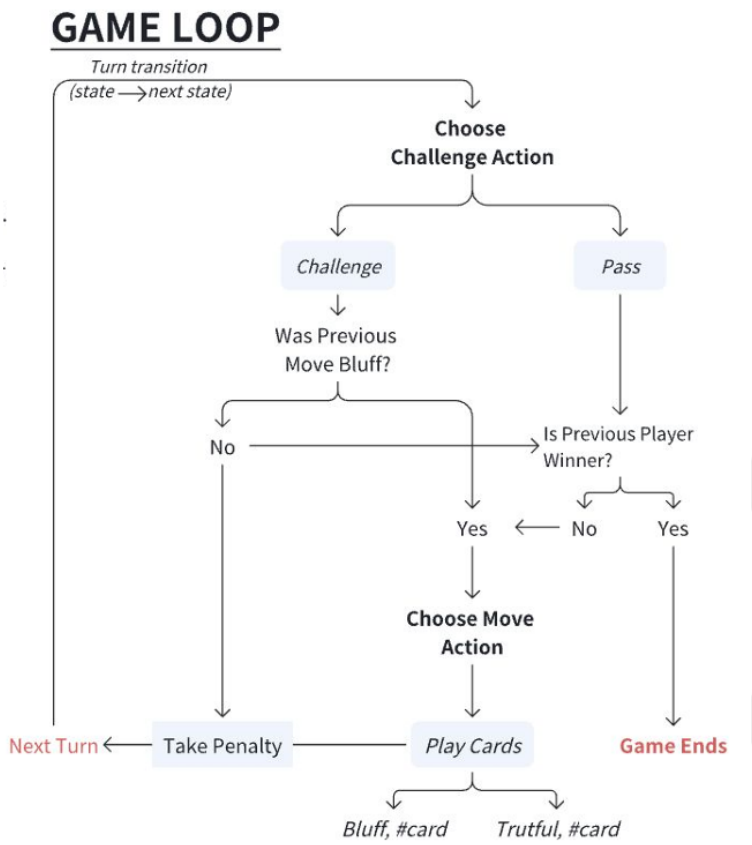

Figure 3.1: Game loop diagram.

Each turn consists of an optional challenge followed by a mandatory move. The player based on turn order begins by deciding whether to challenge the previous move. If no challenge is made, the player proceeds to play. If a challenge is initiated, the previously played cards are revealed. If the revealed move is found to be a bluff, the previous player must collect all cards from the pile, and the challenger continues their turn by making a move. Otherwise, the challenger collects the pile and is not allowed to make a move for that turn. If the challenge has not resolved as a failure, the player performs a move by placing one to six cards face-down and declaring that they match the fixed table color. The move may be truthful or a bluff. Once the move is complete, the turn proceeds to the next player in clockwise order. Once a player empties their hands, the game is concluded and the winner becomes the player who has no cards.

3.4 Mathematical Representation of the Game Elements

In this section, the formulation of game elements, actions, and progression of the modified bluff game are demonstrated. As it is mentioned before, the game is played with classical playing cards where each player receives a number of cards at the beginning of the game. They interact with each other by making moves, announcing the characteristics of those moves, and challenging the other player's move who played before them.

3.4.1 Game Elements and Mechanics

To represent these interactions mathematically, we start with defining fundamental game elements: Player, Deck, Hand and Turn Order. Next, we introduce turn-specific events which means the available set of actions: Moves, Challenges. Then, we define the Card Pile, which reflects the cumulative outcomes of actions during the game.

3.4.1.1 Players

Players can be defined as a finite ordered set where each player has a fixed position. This makes sense, because once the players sit down at a table, their seating arrangement is considered fixed throughout the game, and with this approach, we can use indexes as players' names and achieve creating a generic representation.

$$P = (P_0, P_1, \ldots, P_{n_P-1}), \quad |P| = n_P \in \mathbb{N} \tag{3.1}$$

where n_P represents the number of players in the game, where n_P must be between 2 and 4 if the game is played with only one deck. kth player in the P is, then

$$\alpha : P \times \{0, \ldots, n_P - 1\} \longrightarrow P, \quad \alpha(P,k) = P_k \tag{3.2}$$

The dealer P_d is selected randomly among the players P defined in (3.1). The dealer starts dealing with the player seated next to them in a clockwise direction. Let U be a uniform random selection function defined on the index set $\{1,2,\ldots,(n_P-1)\}$:

$$U : \{0,1,2,\ldots,(n_P-1)\} \longrightarrow \{0,1,2,\ldots,(n_P-1)\}, \tag{3.3}$$

such that for any index j the output $U(j)$ is equally likely to be any element in $\{0,1,2,\ldots,(n_P-1)\}$:

$$\Pr(U(j)=i)=\frac{1}{n_P},\quad \forall k\in\{0,1,2,\ldots,(n_P-1)\}. \tag{3.4}$$

Then, the dealer is defined as:

$$P_d = P_{U(j)}, \tag{3.5}$$

where j is any fixed index (since the distribution of $U(j)$ does not depend on j). Consequently, each player has an equal probability of being chosen as the dealer:

$$\Pr(P_d=P_i)=\frac{1}{n_P},\quad \forall i\in\{0,1,2,\ldots,(n_P-1)\}, \tag{3.6}$$

ensuring that:

$$\sum_{i=0}^{n_P-1}\Pr(P_d=P_i)=1. \tag{3.7}$$

The player who is the dealer gives the first card, that is, the player immediately to their right in the clockwise direction, will be the first to make a move after the distribution process is complete.

$$P'_{\text{first}} = P_{((d+1)\bmod n_P)} \tag{3.8}$$

Let λ be the modular arithmetic function that orders players in cyclic form in a set of players based on the dealer. Let i be the distribution index ranges 0 to (n_P-1), then we can write the player's order by incrementing the i values.

$$\lambda : P \longrightarrow P' \implies \lambda(P)=P' \tag{3.9}$$

$$\lambda(i)=(\text{first}+i)\bmod n_P \tag{3.10}$$

Thus, each player can be described as:

$$P'=\left(P_{\lambda(i)}\right)_{i=0}^{n_P-1} \tag{3.11}$$

For example, suppose there are 4 players in the game. Then, $P=(P_0,P_1,P_2,P_3)$ the dealer is selected based on uniformly random selection as $P_d=P_2$. Then the first player

is $P'_{\text{first}} = P_{((2+1) \bmod 4)} = P_3$. Then the players' order are $P' = (P_3, P_0, P_1, P_2)$ based on the (3.11). So, P_2 is equal to $P_{\lambda(3)}$.

In general, the i-th player in the new order P' (i.e., P'_i) is the same as player P_k in the original set P, where $k = \lambda(i)$ is determined by the modular function.

$$P_k = P_{\lambda(i)} = P'_i \tag{3.12}$$

Furthermore, we define the opponents, O_k as the set of all players in P except the player P_k itself. Then, the opponent set O_k can be defined as:

$$O_k = P \setminus \{P_k\} \tag{3.13}$$

Alternatively, using set comprehension:

$$O_k = \{P_j \in P \mid P_j \neq P_k\} \tag{3.14}$$

3.4.1.2 Deck

The deck of cards consists of a number of distinct cards that are obtained by combinations of one suit and one rank. The set of suits and ranks is defined as follows

$$S = (s_0, s_1, \ldots, s_{n_s-1}) \tag{3.15}$$

For the modified Bluff Game, $n_s = 6$.

$$R = (r_0, r_1, \ldots, r_{n_r-1}) \tag{3.16}$$

For the modified Bluff Game, $n_r = 4$. Then, we can define a card as follows

$$c_l = (s_i, r_j), \quad \text{where } s_i \in S \text{ and } r_j \in R \tag{3.17}$$

e.g., $c_{0,1} = s_0 \times r_1 = (s_1, r_1)$. Thus, we can derive the deck C as the Cartesian product of the suit set S and the rank set R.

$$C = S \times R = (c_0, c_1, \ldots, c_{(n_C-1)}) \tag{3.18}$$

The index $\ell = (0, 1, 2, \ldots, (n_C - 1))$ maps each card to a unique combination of one suit and one rank, so $c_l = c_{ij}$.

The total number of cards n_C is the product of the number of suits n_s and ranks n_r.

$$n_C = |S| \cdot |R| = n_s \cdot n_r \tag{3.19}$$

For a standard deck, $n_s = 6$ and $n_r = 4$, so $n_C = 24$.

Before the players start the game, the deck is shuffled. Since only the index of elements in the deck is changed, we can describe the shuffle with a function σ as follows

$$\sigma : C \longrightarrow C', \quad \text{where } C' = (c_{\sigma(1)}, c_{\sigma(2)}, \ldots, c_{\sigma(n)}) \tag{3.20}$$

where σ is a permutation function as

$$\sigma : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}, \quad \text{such that } C' = \sigma(C).$$

Thus, while the elements in the deck do not change and the size of the deck remains the same, the index of elements is changed:

$$c_\ell = c_{\sigma(\ell)} \quad \forall \ell \in \{1, 2, \ldots, n\}, \quad \text{and} \quad |C'| = |C|. \tag{3.21}$$

After the deck is shuffled, the dealer distributes the cards to the player.

$$\beta : c_i \in C \to h_k, \quad k = i \bmod |P| \tag{3.22}$$

Each card c_i is assigned to player k such that k is the turn order modulo the number of players. By following this method, the cards can be distributed among players in a cyclic manner such that each player receives either an equal number of cards or, at most, one card more, ensuring a nearly even distribution.

3.4.1.3 Table color

The table color is randomly chosen from the card suits in the deck.

$$s_{\text{table}} = U(S) \tag{3.23}$$

where S is the set of suits (colors) given in the equation (3.15) and U is the function that represents the uniform random sampling.

3.4.1.4 Hand

"Hand" represents a set of cards that a player has as a result of dealing. We can define a set of hands as a subset of the deck, and kth player's hand as a proper subset of the deck respectively as given in the equations below

$$H = (h_0, h_1, \ldots, h_{n_H-1}), \quad n_H = n_P, \quad H \subseteq C \tag{3.24}$$

$$\mathscr{H} : (H, P_k) \longrightarrow h_k = (c_0, c_1, \ldots, c_{n_k}), \quad h_k \subset C \tag{3.25}$$

where n_k is the number of cards that kth player holds in its hand. All cards in the deck are evenly distributed among the players. In some cases, cards may not deal evenly; however, the players' hands should not differ by more than one card. The generic equation of n_k in the equation (3.25) that obeys this constraint is given below

$$n_k = \begin{cases} \frac{n_C}{n_P}, & \text{if } n_C \bmod n_P = 0, \text{ for all players,} \\ \frac{n_C}{n_P}, & \text{if } n_C \bmod n_P = k, \text{ for } n_P - k \text{ players,} \\ \frac{n_C}{n_P} + 1, & \text{if } n_C \bmod n_P = k, \text{ for } k \text{ players.} \end{cases} \tag{3.26}$$

Also, the dealer distributes the cards to the players one by one. The cards for the ith player whose receiving order is k are shown as follows:

$$C \cap h_k^t = \{c_\ell\} \iff \ell \bmod n_P = i \iff P_k = P_i') \tag{3.27}$$

Players' hands change based on their actions in a turn, so we express hand H as a function of turn. Let t be the turn, then

$$H(t) = (h_0, h_1, \ldots, h_{n_H}, t), \quad n_H = n_P, \quad H \subseteq C \tag{3.28}$$

Then, we can show the cards that are held by player k as below

$$\mathscr{H} : (H, P_k, t) \longrightarrow h_k^t = (c_0, c_1, \ldots, c_{n_k^t}), \quad h_k^t \subset C \tag{3.29}$$

We can classify each card in a player's hand into matched or unmatched cards based on whether it's suit color is the table color.

$$c^t k, \text{mt.} = \{c_j | c_j \in h_k^s, s(c_j) \in s_{\text{table}}\} \tag{3.30}$$

$$c_{k,\text{unm.}}^t = \{c_j | c_j \in h_k^s, s(c_j) \notin s_{\text{table}}\} \tag{3.31}$$

3.4.1.5 Challenge action

The player can choose to declare challenge to or not to the previous player's move.

$$a_c^t = \delta_k^c(\text{state}) \tag{3.32}$$

where δ_k^c represents the strategy that belongs to the player P_k and a_c^t is the selected action based on the state of the game.

Let l^t be the set of possible challenge actions in turn t which can be shown as follows.

$$l^t = ([1,0],[0,1]) \tag{3.33}$$

where $a_c^t = [1,0]$ represents the declaration of a challenge and $a_c^t = [0,1]$ no challenge is made.

3.4.1.6 Move action

The player can choose to either make a move or challenge the previous player's move. If the player chooses to make a move, they must decide on two components: the move type and the number of cards to play. Let x^t and e^t be the selected type of move and the vector of selected numbers.

$$a_m^t = (x^t, e^t) = \delta_k^m(\text{state}) \tag{3.34}$$

where δ_k^m is the move strategy function for the player P_k, X^t is the set of possible move types at turn t, and E^t is the set of possible card declarations.

The set of possible move types is defined as:

$$X^t = ([1,0],[0,1]) \tag{3.35}$$

where $[1,0]$ represents a bluff move and $[0,1]$ represents a truthful move.

The set of possible declaration vectors is defined as:

$$E^t = (e_0, e_1, \ldots, e_{n_S-1}) \tag{3.36}$$

where e_i is a unit vector with a dimension as n_S, in other words number of suits in the deck (3.15). So, we can write e_i as given below,

$$e_i = e_{i,j} = [\delta_{i,j} | j = 0, 1, .., (n_S - 1)] \tag{3.37}$$

where $\delta_{i,j}$ is a Kronecker-Delta function which yields 0 if $i \neq j$ and 1 if $i = j$. If n_i is the chosen vector, then $i+1$ is the number of cards chosen to play in turn t.

3.4.1.7 Card pile

"Card Pile" refers to the collection of played cards which gathered into the center in the real-life gameplay. The card pile updates based on actions taken.

$$Y^t = (y_1, y_2, \ldots, y_{n_Y^t}), \quad Y \subset C \text{ and } y_i(t) = m^t \tag{3.38}$$

The observed card pile consists of announced (declared) cards:

$$Y^t_{\text{observed}} = (y_1, y_2, \ldots, y_{n_Y^t}), \quad y_i = (s_{\text{table}}, |m^t|) \forall t \tag{3.39}$$

The relative pile refers to the collection of cards as a frequency that a player has played during the game.

$$Y^t_k = (y_1, y_2, \ldots, y_{n_Y^t}), \quad Y \subset C \tag{3.40}$$

where $y_i(t) = F^t$ which is card frequency of a move if $k = t \mod n_P$ Here, Y_k^t represents the relative pile for player k at time t, consisting of n_Y^t moves made by player k, where each $y_i(t)$ is a move frequency f_k^t made by player k at time t.

$$Y^t = \{Y_k^t | \forall k\} \tag{3.41}$$

The total relative pile Y_k^t at time t is the union of the relative piles for all players, where each player k has a corresponding relative pile Y_k^t. We can show how many cards that played by kth players is matched card as given below

$$Y_{k,mt.}^t = \sum_{c \in Y_k^t} f(r)\{r(c) = s_{\text{table}}\} \tag{3.42}$$

3.5 Game setup

This section outlines the formal structure of the bluff-based card game and we introduce the detail of the complete gameplay pipeline from initialization to termination. It begins with the game setup which consist of steps as follow: deck shuffling, card distribution, and initial state configuration. The game state is defined as a comprehensive representation of game observation which includes both perfect information which is information that is known for all players, and player-specific information. The turn phase is divided into two key action phases: the challenge phase, where players may dispute the validity of the previous move, and the move phase, where players make their strategic plays and declarations. After that, we define state transitions and variable updates for each turn transation. Finally, the section defines the termination condition for the game and the criteria for determining the winner. Together, these components formalize the dynamic decision-making process and cyclic flow of the game.

3.5.1 Preparation phase

At the beginning of each game, the card deck is shuffled to ensure randomness. We define a shuffling operation using a permutation function σ over the index set $1, 2, \ldots, n_C$, where n_C is the total number of cards. Applying this permutation, we obtain a shuffled deck $C' = \sigma(C) = \left(c_{\sigma(1)}, c_{\sigma(2)}, \ldots, c_{\sigma(n_C)}\right)$.

Following the shuffling process, cards are distributed sequentially to the set of players $P = (P_1, P_2, \ldots, P_{n_P})$, where n_P is the total number of players. Each player P_k receives a hand $h_k \subset C'$ defined by:

$$h_k = \{c_\ell \in C' \mid \ell \bmod n_P = i \text{ and } (d+i) \bmod n_P = k\}, \tag{3.43}$$

where d is the index of the dealer. This distribution ensures that the number of cards held by any two players differs by at most one, satisfying the condition:

$$\Big| |h_k| - |h_{k'}| \Big| \leq 1, \quad \forall k, k' \in \{1, \ldots, n_P\}. \tag{3.44}$$

After the cards have been dealt, the game state is initialized. The pile of played cards is initially empty, denoted by $Y(0) = \emptyset$, and the set of publicly observed declarations is also initialized as $Y\_\text{observed}(0) = \emptyset$. A dealer $P\_d \in P$ is chosen uniformly at random, and the first player to act is determined as:

$$P_{\text{first}} = P_{(d+1) \bmod n_P}. \tag{3.45}$$

Finally, the turn counter is initialized as $t \leftarrow 0$.

3.5.2 Game state

The state represents all the essential information required to make decisions at a given point in the game. Player strategies use this set of information when making action decisions. Let T be the trajectory of states and s^t be the state at turn t.

$$T = (s^0, s^1, \ldots, s^t) \tag{3.46}$$

The players decide their action based on information about player's and opponents' hands and, card pile. For a player k who plays at turn t, then index of the previous player and next player can be defined respectively as below

$$p = k - 1 \bmod n_P, \quad n = k + 1 \bmod n_P \tag{3.47}$$

the game state can be written as numerical information about card types based on where they are placed as follows

$$s^t = (t, k, |h_k^t|, |c_{k,\text{mt.}}^t|, |c_{k,\text{unm.}}^t|, |h_p|, |h_n|, |h_{O_k}||Y^t|, |Y_k^t|, |Y_{k,mt.}^t, |d^{t-1}) \tag{3.48}$$

where t is the turn number, k is the player index who is playing at turn t, $|h_k^t|$ is the number of cards in the player's hand, $|c_{k,\text{mt.}}^t|$ is the number of matched cards in the player's hand, $|c_{k,\text{unm.}}^t|$ is the number of unmatched cards in the player's hand, $|h_p|$ is the number of cards in the previous player's hand, $|h_n|$ is the number of cards in the next player's hand, $|h_{O_k}|$ is the number of cards in opponents' hands, $|Y^t|$ is the number of cards in the pile, $|Y_k^t|$ is the number of cards in the pile that were played by the player, $|Y_{k,mt.}^t$ is the number of matched cards in the pile that were played by the player, and $|d^{t-1}$ is the last declaration.

3.5.3 Turn phase

The turn phase describes states of the game in which players, in turn order, perform actions that progress the game. In this section, we introduce move and challenge phases with respect to gameplay rules and define state, turn transition, turn history for the cyclic progression of the gameplay.

The turn order is determined before the first move as follows: For all n_k values, the player acting in the first turn "$t = 0$" is the player who is on the dealer's right as in equation (3.8). Turns progress clockwise by default; that means the player to the first player's right is the next player.

Let t be the turn number and P_{first} is the first player we defined previously then whose player's turn is the turn t can be represented as

$$\mathscr{T} : (P, t) \longrightarrow P_k \quad \text{where } k = (\text{first} + t) \bmod n_P \tag{3.49}$$

The first player starts the game by making a move by selecting a set of cards and announces the characteristics of those cards (rank and number of cards) to their opponents. For each turn, except for the first and the one with an empty pile, players

must decide whether to challenge regarding the previous move before making their own move. For general expression, we define action phases as challenge and move.

Turn starts with challenge phase. If the pile is non-empty and the move is non-empty, then the player can challenge; otherwise, they have to pass to challenge since there is no card to claim it is a bluff. Thus, let b^t be the performed challenge action, which is the action that is selected by the player's strategy if the condition satisfies in the current state; otherwise, we label $[0,0]$ which describes mandatory not to challenge. When a challenge occurs, the face-down cards m^{t-1} played by the previous player P_p are turned over.

$$b^t = \begin{cases} a_c^t = \delta_k^c(s^t), & \text{if } |Y^t| \neq \emptyset \text{ and } d^{t-1} \neq 0, \\ [0,0], & \text{otherwise.} \end{cases} \tag{3.50}$$

The resolution of a challenge is determined by the current player's decision b^t and the actual bluff status of previous player's last move. We can define the challenge action outcome R^t as follows:

$$R(b^t) = \begin{cases} [1,0], & \text{if } b^t = [1,0] \text{ and } d^{t\,\text{prev}} = f(s_{\text{table}}, m^{t-1}), \\ [0,1], & \text{if } b^t = [1,0] \text{ and } d^{t\,\text{prev}} \neq f(s_{\text{table}}, m^{t-1}), \\ [0,0], & \text{otherwise.} \end{cases} \tag{3.51}$$

Based on the challenge action outcome, card pile and hand of players are updated. According to our definition h_p and h_k are the sets of cards held by the previous and current players respectively and we can show their updated version as below

$$h_p \leftarrow \begin{cases} h_p \cup Y^t, & \text{if } R^t = [1,0], \\ h_p, & \text{otherwise.} \end{cases} \tag{3.52}$$

$$h_k \leftarrow \begin{cases} h_k \cup Y^t, & \text{if } R^t = [0,1], \\ h_k, & \text{otherwise.} \end{cases} \tag{3.53}$$

If a challenge occurs, the card pile is cleared:

$$Y^{t\prime}, Y_{observed}^{t\,\prime}, Y_k^{t\prime} \leftarrow \begin{cases} \emptyset, & \text{if } b^t = [1,0], \\ Y^t, Y_{observed}^t, Y_k^t & \text{otherwise.} \end{cases} \tag{3.54}$$

Finally, turn state is updated for the next stage of the turn,

$$s^{t\prime} \leftarrow s^t \tag{3.55}$$

After the challenge phase ends, the move phase starts. If the kth player had a successful bluff resolution or passed challenging the previous player, then they can perform their decided move. In other words, if the player did not fail their challenge attempt, then they can make a move based on their strategy; otherwise, they cannot play their cards as shown below.

$$\mu_k^t = \begin{cases} a_m^t = \delta_k^m(s^t), & \text{if } R(b^t) \neq [0,1], \\ ([0,0], e_{default}^t), & \text{otherwise.} \end{cases} \tag{3.56}$$

where $e_{default}^t$ is a tuple (finite ordered set) as $[0,0,0,0,0,0]$. According to the chosen move type (3.35), the available cards to play are selected:

$$c_{k,\text{chosen}}^t = \begin{cases} c_{k,\text{mt.}}^t & \text{if } x^t = [0,1], \\ c_{k,\text{unm.}}^t & \text{if } x^t = [1,0]. \end{cases} \tag{3.57}$$

The number of cards to play is determined by the following:

$$n = \min(i+1,\ |c_{k,\text{chosen}}^t|) \tag{3.58}$$

where i is index of chosen declaration vector e_i if $R^t \neq 0$ in (3.37) else $e_i = [0,0,0,0,0,0]$. Finally, the actual move is created by randomly sampling n cards from the chosen set of cards, since suit types do not affect the declaration.

$$m^t = \text{sample}(c_{k,\text{chosen}}^t, n) \tag{3.59}$$

where $m^t = \{c_1, c_2, \ldots, c_{n_S}\}$. Thus, the declaration is

$$d^t = n \tag{3.60}$$

the color frequency of the move can also be written as:

$$f(s) = \sum_{c \in m^t} \mathbf{1}\{s(c) = s\} \tag{3.61}$$

$$F^t = \{s_i : f(s_i) | \forall s_i \in m^t\} \tag{3.62}$$

for example if $m^t = \{(1,2), (2,1), (2,2)\}$, then $F^t = \{1 : f(s=1), 2 : f(s=2)\}$ where $f(1) = 1$ and $f(2) = 2$.

After the action is performed card pile and the current player's hand are updated since card transaction occurs. Thus, h_k, updated when a non-empty move is made by kth player as below:

$$h_k^t \leftarrow h_k^t \setminus m^t. \tag{3.63}$$

and cards in m^t added into the card pile,

$$Y^{t\prime\prime} \leftarrow Y^{t\prime} \cup m^t. \tag{3.64}$$

The declaration of move action is added to the observed pile of declared information:

$$Y_{observed}^{t}{}^{\prime\prime} \leftarrow Y_{observed}^{t}{}^{\prime} \cup d^t, \tag{3.65}$$

$$Y_k^{t\prime\prime} \leftarrow Y_k^{t\prime} \cup m^t. \tag{3.66}$$

and finally, state is updated since player's hand and piles changed,

$$s^{t\prime\prime} \leftarrow s^{t\prime} \tag{3.67}$$

The outcome of a bluff move is determined by the next player's decision b^{t+1} and the actual bluff status of kth player's move. We can define the bluff outcome V^t as follows:

$$V(m^t) = \begin{cases} [1,0], & \text{if } m^t = [1,0] \text{ and } m^t = [0,1], \\ [0,1], & \text{if } m^t = [1,0] \text{ and } m^t = [1,0], \\ [0,0], & \text{if } m^t = [1,0]. \end{cases} \tag{3.68}$$

where $[1,0]$ is a successful bluff, $[0,1]$ means an unsuccessful bluff that is caught by the next player.

The game continues until a player has no cards left. We evaluate the Boolean value isPlayerHandEmpty each turn, t. We check this condition to see if the game is over as follows:

$$\text{isGameEnd} = \begin{cases} 1, & |h_p^t| = 0 \text{ and } R^t \neq [1,0] \\ 0, & \text{otherwise.} \end{cases} \tag{3.69}$$

Suppose kth player empties their hand with move m^t at turn t. Then kth player wins which evaluated with function ω, and no turns remain, as shown below:

$$\text{isGameEnd} = 1 \implies \omega(P,t) \longrightarrow P_p \tag{3.70}$$

Then, we can set the winner of the game and losers as follow:

$$w_k^t = \begin{cases} [0,0] & \text{if isGameEnd} = 0 \\ [0,1] & \text{if isGameEnd} = 1 \text{ and } \omega(P,t) \neq P_k \\ [1,0] & \text{if isGameEnd} = 1 \text{ and } \omega(P,t) = P_k \end{cases} \tag{3.71}$$

3.5.3.1 Turn transition

After the player performed all of the available actions that belongs to them, the previous player, current turn index, current player updated.

$$P_p \leftarrow P_k \tag{3.72}$$

$$t \leftarrow t+1 \tag{3.73}$$

by using the new turn t,

$$k = (\text{first} + t) \bmod n_P \tag{3.74}$$

and new state for the new turn is constructed with pile information from the last state that we get from the end of the move phase, new current player's characteristics of cards and new turn index.

$$s^t \leftarrow s^{t\prime\prime} \tag{3.75}$$

3.5.3.2 Turn history

Monitoring player actions during the game is crucial, as this information guides the decision-making process since actions and their outcomes change the game state. In this section, we present game records that include both public and private information.To accomplish this, first define a set that includes actions and their outcomes; then create a set of turn records that is updated with each new turn associated with player k.

$$\tau^t = \{k, b^t, R(b^t), (\mu_k^t)_x, (\mu_k^t)_e, m^t, d^t\} \tag{3.76}$$

$$\tau = \{\tau^t \mid t = 0, 1, \ldots, t_{end}\} \tag{3.77}$$

4. METHODOLOGY

This chapter presents the methodological framework used to develop, simulate, and evaluate agent strategies in the bluff-based card game environment. We introduce the design and decision-making mechanisms of baseline, learning-based, and reasoning-based agents, detail the implementation of the simulation environment, define the training configurations for reinforcement learning agents, and explain the opponent filtering procedure applied for evaluating the LLM agent. Finally, we describe the performance metrics used to compare agent behaviors under varying game conditions.

4.1 Agent Models

This section introduces the architectures and decision-making mechanisms of the agent models used in the bluff-based card game. Each agent is designed to independently evaluate and select actions for two distinct action types, which are challenge and move, based on its internal strategy, which may rely on random choice, short-term probabilistic reasoning using current state information, statistical inference from limited observations, reinforcement learning, or language model reasoning. Agents are designed to perform decision making for each action type, as shown in Table 4.1 below.

Table 4.1: Action decision logic of the DQN agent.

Type	Decision Made
Challenge	Challenge vs. Not
Move	Truth vs. Bluff, Amount

4.1.1 Decision-making strategies of baseline agents

This section introduces the core strategy models for agents which are described as rule based random selection used to guide player decision-making within the game. Each strategy defines a distinct approach for choosing actions based on available information.

We categorize three main baseline strategies: Random strategy where a uniform random policy is performed and selects actions without considering any contextual

information, purely change-based. State-Dependent Strategy which is a conditional random policy that uses current game observations to determine challenge and bluff decisions. Lastly, Bayesian-State Mixed Strategy where a hierarchical approach is employed as combination of state-dependent observations and Bayesian priors derived from previous game outcomes to adaptively adjust challenge and bluff probabilities. So, each strategy operates over the same action space but leverages different levels of game knowledge and inference to select actions. In what follows, we mathematically define these strategies and describe the corresponding decision rules.

4.1.1.1 Agent with random strategy

We use the Random Agent as a benchmark baseline to evaluate whether other agents perform better than chance-level behavior.

In a random strategy, the player chooses actions randomly. Let $\mathrm{U}(A)$ be a uniform distribution over set A, indicating equal probability for all elements.

Let:

$$Z \sim \mathrm{U}(A). \tag{4.1}$$

Z be a discrete random variable representing the chosen action. A uniform distribution describes a scenario where each element has the same probability of selection due to random and uniform choice. Thus, the probability of selecting action a_i is:

$$P(Z = a_i) = \begin{cases} \frac{1}{|A|}, & \text{if } a_i \in A, \\ 0, & \text{otherwise.} \end{cases} \tag{4.2}$$

Then, the player decides whether to challenge or not by employing a random strategy $\delta_c^t\ U(A)$ as:

$$a_c^t \longleftarrow \delta_k^c(\text{state}) \quad \text{where } P(Z = a_c^t) = \frac{1}{|l^t|} \tag{4.3}$$

and selects their move as follows:

$$a_m^t \longleftarrow \delta_m^t(\text{state}) \tag{4.4}$$

$$e^t \longleftarrow \delta_m^t(\text{state}), \quad \text{where } P(Z = x^t) = \frac{1}{|X^t|} \tag{4.5}$$

$$e^t \longleftarrow \delta_m^t(\text{state}), \quad \text{where } P(Z = e^t) = \frac{1}{|E^t|} \tag{4.6}$$

4.1.1.2 Agent with state-dependent strategy

The conditional random strategy utilizes a probabilistic method to determine an action by comparing it to a predefined threshold. However, the player does not think about the previous player's past actions and challenges that they received and focus only on the current observables about the cards.

To do that, the strategy uses the current game observations that are given in the state. Basically, this strategy consists of two steps for the challenge action, such as counting available matched cards that can be in both opponent's hand or pile, and evaluating the probability of the last declared amount of matched cards in the previous player's hand at the previous turn $t-1$. Moreover, for the move action, a declaration amount is determined based on upper limit for the card amount, then the player evaluates probability of having any matched cards in their hands with respect to next player's point of view and based on this probability they bluff or not. Then, according to the chosen move type, the player makes their move. If the player has no matched cards, the probability of choosing bluff is 1 and vice versa. Let us define several quantities as the number of matched cards that may in opponents hands and number of cards played by opponents that are used to make estimations as follows:

$$C^t_{O_k,mt.} = n_S - (C^t_{k,mt.} + |Y^t_{k,mt.}|) \tag{4.7}$$

$$|Y^t_{O_k}| = |Y^t| - |Y^t_k| \tag{4.8}$$

$$|C^t_{opp}| = |Y^t_{O_k}| + |h^t_{opp}| \tag{4.9}$$

$$|h^{t-1}_p| = |h^t_p| + d^{t-1} \tag{4.10}$$

$$UL = \begin{cases} |C^t_{k,mt.}| & \text{if } |C^t_{k,mt.}| > 0, \\ \min(h^t_k, n_S) & \text{otherwise.} \end{cases} \tag{4.11}$$

Let us define a β function that evaluates probability of having cards based on given parameters.

$$\beta(a,b,c,d) = \frac{\binom{a}{b} \cdot \binom{c-a}{d-b}}{\binom{c}{d}} \tag{4.12}$$

Let q_c^t be the estimation of probability of not having declared cards in previous player's hand and p_c^t be the probability of choosing performing the challenge declaration then process for selecting an challenge action as given below:

$$q_c^t = \begin{cases} 1, & \text{if } C^t_{O_k,mt.} < d^{t-1} \\ 1-\beta(|C^t_{O_k,mt.}|, |d^{t-1}|, |C^t_{opp}|, |h_p^{t-1}|), & \text{otherwise.} \end{cases} \tag{4.13}$$

by using q_c^t, we determine probability of declaring challenge is as follow:

$$p_c^t = \begin{cases} 0, & \text{if } (|h_k^t| \leq 2 \text{ and } q_c^t < 0.8) \text{ or } q_c^t = 0, \\ 1, & \text{if } (|h_p^t| == 0 \text{ and } |h_k^t| > \frac{|h^t_{O_k}|}{|O_k|}) \text{ or } q_c^t = 1, \\ 0.8, & \text{if } |h_p^t| == 0 \text{ and } |h_k^t| \leq \frac{|h^t_{O_k}|}{|O_k|}, \\ q_c^t, & \text{otherwise.} \end{cases} \tag{4.14}$$

Then the challenge action is selecting as follow:

$$a_c^t = \begin{cases} 0, & \text{with probability } p_c^t \\ 1, & \text{with probability } 1-p_c^t \end{cases} \tag{4.15}$$

The process of selecting the move action consists of 3 steps such as choosing a declaration which is shown as e^t, evaluating q_m^t probability of having no matched cards for estimation of the occurrence of challenge for selecting the declaration and the probability of playing the bluff as follows:

$$e^t_{default} \rightarrow e_i^t \quad \text{where } i \sim U\{1, UL\} \tag{4.16}$$

since $e^t_{default} = (0,0,0,0,0,0)$ and $e_i = e_{i,j} = (\delta_{i,j} | j = 0,1,..,(n_S-1))$.

Let C_{O_n} be the opponents cards with respect to next player, then we can write corresponding equation for C_{O_n} as

$$|C_{O_n}| = n_S \cdot n_R - |h_n^t| \tag{4.17}$$

$$q_m^t = \begin{cases} 1, & \text{if } |C^t_{k,mt.}| = 0 \\ 0, & \text{if } |h_k^t| - |C^t_{k,mt.}| = 0 \\ \beta((|C_{O_n}| - |C^t_{k,mt.}|), i, |C_{O_n}|, |h_k^t|), & \text{otherwise.} \end{cases} \tag{4.18}$$

Then we can write the probability of playing bluff as

$$p_m^t = \begin{cases} 1, & \text{if } q_m^t > 0.8 \\ q_m^t, & \text{if } 0.5 < q_m^t \leq 0.8, \\ 0.5, & \text{if } 0.3 < q_m^t \leq 0.5, \\ 0, & \text{otherwise.} \end{cases} \tag{4.19}$$

Then the move action is selecting as follow:

$$a_m^t = \begin{cases} (0, e_i), & \text{with probability } p_m^t \\ (1, e_i), & \text{with probability } 1 - p_m^t \end{cases} \tag{4.20}$$

4.1.1.3 Agent with Bayesian-based strategy

In this section, we formalize the mathematical derivation used for computing the probability of initiating a challenge and bluff based on Bayesian reasoning. Hierarchical formulation reflects an opinion as confidence in decision-making based on both empirical priors and current belief estimates.

The challenge decision considers both the estimated success of the challenge itself and the prior behavior of the previous player.

Let q_c^t denote the estimated probability that the declared move by the previous player is bluff as given (4.13). This value is computed from the observed cards and estimated likelihood, as defined in Section 4.1.1.2. Additionally, let $\Pi_p^{K=(\mu^t)_x}$ be the Bayesian probability that previous player played a bluff move in previous games. This corresponds to the Bayes output of the bluff action model.

We then define the joint Bayesian probability p_c^t of initiating a challenge using a modified noisy-OR combination:

$$p_c^t = \begin{cases} 0.1, & \text{if } q_c^t \cdot \Pi_p^{K=(\mu^t)_x} = 0 \\ 0.9, & \text{if } q_c^t \cdot \Pi_p^{K=(\mu^t)_x} = 1 \\ \dfrac{q_c^t \cdot \Pi_p^{K=(\mu^t)_x}}{q_c^t \cdot \Pi_p^{K=(\mu^t)_x} + (1 - q_c^t) \cdot (1 - \Pi_p^{K=(\mu^t)_x})}, & \text{otherwise} \end{cases} \tag{4.21}$$

This structure follows a generalized Bayesian update mechanism with prior and likelihood adjusted to avoid extremal effects when either component is 0 or 1.

Furthermore, with Bayes' formula, the strategy ensures even if the turn-based calculations show a low probability of occurring bluff, the player still can challenge with a higher probability.

Additionally, we define the final challenge action probability $\hat{p}_c^t$ by refining p_c^t using resolution of player's challenge action. Let r_k be the Bayesian resolution probability for player k, that is, the posterior success probability of the player's past challenge actions:

$$r_k = \Pi_k^{K=R(b^t)} \tag{4.22}$$

thus, r^k shows bayesian estimation of getting $[1,0]$ after declaring a challenge.

Then the adjusted challenge probability is given by:

$$\hat{p}_c^t = \begin{cases} 1, & \text{if } |h_p^t| = 0 \text{ and } p_c^t > 0.8 \text{ and } h_k^t > \dfrac{|h_{O_k}^t|}{|O_k|} \\ 0.8, & \text{if } |h_p^t| = 0 \text{ and } h_k^t \leq \dfrac{|h_{O_k}^t|}{|O_k|} \\ p_c^t, & \text{if } r_k > 0.6 \\ p_c^t \cdot r_k, & \text{if } 0.1 < r_k \leq 0.6 \\ 0, & \text{otherwise} \end{cases} \tag{4.23}$$

So, according to the final probability to decide declaring a challenge to previous move, the player who uses Bayesian strategy never challenges the opponent's action when the success probability is less than %10 and reduces the probability of declaring challenge as a success rate if the success is less than %60. So, the player takes low risk when past game resolutions were not enough efficient.

The bluff decision relies on both the player's hand composition at that turn and the expected likelihood of being challenged by the next player. Let q_m^t be the estimated probability that the player can successfully bluff, as derived in Equation (4.18). Additionally, let $\Pi_n^{K=b^t}$ denote the Bayesian probability that the next player n will challenge a move, based on their historical challenge behavior.

The bluff decision begins with two deterministic checks: If the player has no matched cards, then $p_m^t = 1$ and if the player's hand only consists of matched cards, then $p_m^t = 0$.

Otherwise, we define an adjusted probability score using both the opponent's challenge behavior and the player's estimated success.

$$p_m^t = \frac{(1 - \Pi_n^{K=b^t}) + q_m^t}{2} \tag{4.24}$$

This formulation assumes that bluffing is more likely to succeed if the opponent is historically less likely to challenge and if the estimated likelihood q_m^t is high.

Finally, the probability of playing a bluff is defined as a threshold-based function of p_m^t:

$$\hat{p}_m^t = \begin{cases} 1, & \text{if } p_m^t > 0.8 \\ p_m^t, & \text{if } 0.5 < p_m^t \leq 0.8 \\ 0.5, & \text{if } 0.4 < p_m^t \leq 0.5 \\ 0, & \text{otherwise} \end{cases} \tag{4.25}$$

4.1.2 Agent with learning-based strategy

The Deep Q-Network (DQN) algorithm is applied to model learning-based (reinforcement learning) agents. In this section, the DQN structure, state representation, reward model, and training process are introduced.

4.1.2.1 Design

DQN Agents are designed to perform actions in the bluff-based card game environment, and for the implementation, we use PyTorch. To implement and train these agents effectively, we use PyTorch with the following key hyperparameters in Table 4.2:

Table 4.2: Hyperparameters used in the DQN agent.

Parameter	Value
epsilon	1.0
epsilon_decay	0.996
gamma	0.99
batch_size	64

The parameter epsilon refers to the initial exploration rate used in the ε-greedy strategy. Epsilon decay represents the multiplicative factor by which the epsilon value is reduced after each episode to gradually shift from exploration to exploitation. Gamma denotes

the discount factor applied when computing the present value of future rewards. Finally, batch size specifies the number of samples drawn from the replay memory during each training update [23], [16].

All tensors and models are moved to GPU (if available) using torch.device("cuda"), ensuring efficient computation during both forward and backward passes [23].

In addition to the agent's learning configuration, the input representation plays a critical role in performance. The input state is,the game state, s^t that we defined in the equation (3.48), is represented as a concatenated vector of scalar features from the game environment, including:

- Player hand size, matched/unmatched card counts
- Opponent hand sizes and total cards that played and hold by opponents
- Current turn index and table rank
- Previous declarations
- Observed pile history

These observations are normalized and flattened into a fixed-size tensor of size 12 for the neural network.

The DQN agent uses a shared backbone consisting of two fully connected layers with 128 ReLU-activated units each. Thus, with this custom implementation, the dual-action structure of the game is created; in other words, separating the estimation of challenge and move actions into two distinct network heads. Since an agent has to decide on challenge or not, then they decide on which move type they are going to play with and how many cards, the backbone is represented by two output heads: a scalar output for the challenge decision (sigmoid) and two outputs for move decisions as a sigmoid output and a softmax output.

For the challenge action, challenge head either randomly chooses to challenge or passes the state through a neural network head that outputs a probability; a value above 0.5 results in issuing a challenge.

For the move action, the agent first decides whether to bluff or play honestly, and then selects a move amount from six options. During exploration, both choices are random and checked for validity. During exploitation, the state is passed through a separate network head that outputs a move type probability and a softmax distribution over move amounts (from 1 to 6) which is number of cards that will played in that move.

The agent uses a replay buffer of size 10,000 transitions, which stores recent gameplay experiences as a sliding window. During training, transitions are stored in a replay buffer and sampled in mini-batches to update the agent's policy by minimizing the Mean Squared Error (MSE) between predicted and target Q-values, following the Bellman equation [24]. Each transition stored in replay memory includes a binary flag $\alpha_t \in \{0,1\}$ indicating the action type: $\alpha_t = 0$ for challenge actions, and $\alpha_t = 1$ for move actions. In this way, independent learning updates for the dual-decision structure of the game is satisfied. Challenge updates are performed by predicting a scalar Q-value from the challenge head and computing the MSE loss against the target network's output. Move updates consist of two components: a binary classification for move type (truth or bluff), and a categorical selection among six possible move amounts, both of which are trained using MSE loss. The total loss is calculated as the sum of the challenge and move losses. Synchronized with those of the policy network and target network, synchronized periodically for stability, following the target network update strategy proposed in the original DQN formulation [16].

Moreover, DQN-Agent maintains an experience replay memory $\mathscr{D}$ with a maximum capacity of 10,000 transitions. Once full, older transitions are overwritten. At each step, the following tuple is stored (convention: $x^t \rightarrow x_t$):

$$(s_t, a_t, \alpha_t, r_t, s_{t+1}, d_t) \tag{4.26}$$

where s_t is the state , a_t the action, $\alpha_t \in \{0,1\}$ the action type (challenge or move), r_t the reward, s_{t+1} the next state, and d_t the episode termination flag.

Agent weights were saved using PyTorch serialization tools at the end of training by labeling with agent group, date and time.

4.1.3 Agent with reasoning-based strategy

4.1.3.1 Development and technical details

We developed a Large Language Model (LLM)-based agent utilizing the GPT-4o model via OpenAI's API. Instead of learning through weight updates, this agent selects actions through prompt-based reasoning. The implementation is modular and consists of interpreters that extract key information from the environment: current observations, opponent behavioral statistics, and recent action memory. These elements are combined to form a structured prompt, which is submitted to the LLM. The model's response is parsed from a predefined JSON schema and executed as an action within the environment.

The LLM agent is used to evaluate whether a pretrained, general-purpose language model can exhibit theory-of-mind and strategic reasoning in an imperfect-information, deception-oriented game environment when game rule set, observation, and probabilistic information are provided with a strategy guidance.

4.1.3.2 Prompt structure

We converted the game rule, history, and current game data into natural tone text format and transferred the information as prompt sets on a scenario basis to the agent.

For token efficiency, game data sets were defined separately for challenge and move actions. In addition, we created different strategy guidance for different action types, which methodologically presents how to evaluate this information and different situations that may arise. The approach follows principles described in OpenAI's Prompting Guide for task decomposition and structured LLM inputs [12].

The agent receives structured input comprising the current observation, game rules, recent history, and strategic guidance. This context is processed through a self-evaluation step within the language model to determine the appropriate challenge or move action as given in Figure 4.1.

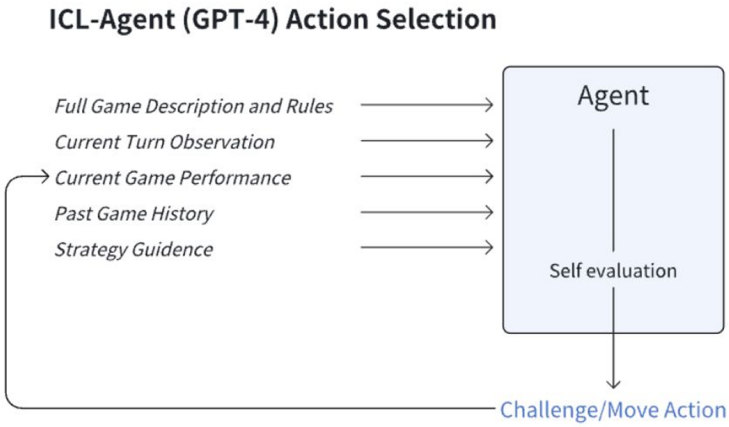

Figure 4.1: LLM agent - Desicion Making Flow: Illustration of the LLM-Agent (GPT-4) action selection mechanism.

These modular components can be described as follows:

- **Game Rule and Definitions**: The prompt section where general game rules are explained.
- **Current Observation:** The state information of the current round is given in this section. So, this is a natural language summary of the current card distribution, last move declaration and turn index.
- **Bayesian Behavior Summary:** Individual (no pooling) probabilities of bluffing, challenging, and their successful outcomes for each player, based on historical behavior and updated each new game. Also, win rate is represented in this part.
- **Memory:** A short history of the agent's and opponents' recent actions, including past bluff outcomes, challenge successes/failures, and bluff patterns.
- **Guidance Template:** A task-specific instruction which shows the model to reason step-by-step to evaluate different action scenarios, make inference about opponents' cards and behavior and return its decision in a strict JSON format.

4.1.3.3 Action selection

The response is generated by the model by following the structure enforced by the guidance section, along with a summary of reasoning.

Two decision types are supported:

- **Challenge Decisions:** The model outputs {"challenge": [1,0]]} to challenge or {"challenge": [0,1]} to pass.
- **Move Decisions:** The model outputs {"move_type": [1,0] for bluff or [0,1] for truthful move, "amount": X}, where amount is the declared number of cards (1–6).

For each game turn, the agent constructs a new prompt incorporating the latest observation, behavior statistics, and memory. It submits this prompt to GPT-4o and receives an action as a JSON response. The decision is then applied within the simulation, and all relevant prompt–response pairs are logged for post-game analysis. This configuration provides significant insight into the LLM agent's reasoning process, enables the targeted exclusion of components, such as memory or Bayesian inputs, for the purpose of ablation studies, and coincides with the recent research on theory-of-mind prompting in language models [4].

4.2 Simulation Environment

In this section, we introduce the environment setup, game flow, and training configurations.

4.2.1 Environment Setup

To run simulations, we implemented a modular environment that allows different agent configurations to interact within the bluff-based card game. Each match begins with environment initialization using a SetUpGame object, that is responsible from agent instantiation, card distribution, and initial game conditions. This environment includes a turn manager, represented by the GenerateTurn class, which provides sequential access to the current state, transitions, and rewards.

The game environment exposes an env.reset() method to initialize the episode and a env.step() method to execute player actions.These two functions comply with established reinforcement learning norms, allowing them to work with learning agents and rule-based agents [23].

During evaluation, no learning or reward computation occurs for the DQN agents execute actions purely based on their pretrained policy networks if it is not evaluation of training them. Similarly, the environment does not store transitions or update replay buffers if it is not training. So, all decisions in final agent evaluation are based on current policy outputs and state observations, not shaped by in-episode feedback.

4.2.2 Game initialization and strategy assignment

At the beginning of each match, an Agent (Players) object is created. This object instantiates the required number of agents and assigns strategies to each one. The strategy assignment is modular, supporting:

- Baseline Strategies (Random, State-Dependent, Bayesian)
- DQN-Based RL Agents
- LLM-based Agent via OpenAI API

Each agent is assigned a unique player ID. Match-specific naming is constructed using the concatenation of these IDs to ensure traceability during logging and evaluation as given in Table 4.3.

Table 4.3: Agent types and configurations used in simulations.

Category	**Agent ID**
Baseline Strategies	Baseline_Random Baseline_State_Dpd Baseline_Bayes
DQN-Based RL Agents	RL_Agent_0 RL_Agent_1 RL_Agent_2 RL_Agent_3
LLM-Based Agent	LLM_Agent

The agents are positioned in rotating orders to eliminate positional advantage. Specifically, agents are assigned to every possible player index (since the game is played by 3 players as given in game design, these possible indexes are 0, 1, 2) in a cyclic manner throughout the simulations, which provides agents equal environmental

conditions and examines their performance under random ordering to get a more generalized performance result: chance to play first, middle, and last positions in the turn order in long-term simulation.

Card distribution is randomized at the beginning of every episode using a uniform shuffling function applied to the deck by using NumPy's random permutation function to ensure stochastic fairness in initial conditions for each agent and preserve statistical consistency.

Simulation sample size (episode/game number) is selected as 2000 for RL Agent trainings, 1000 for each match-up except for ones with the LLM-Agent which are run for 250 episodes due to API cost and latency constraints. Although the LLM-agent involved episodes are less than other match ups, since we evaluate multiple scenarios we still have enough results to compare performance metrics. This simulation framework creates an environment that ensures metric convergence and statistical significance across key performance indicators such as bluff and challenge success rates, win rates, and session lengths.

4.2.3 Game loop and execution flow

The simulation progresses through a centralized game loop, implemented in main(). Each episode consists of the following sequence:

1. Game state is initialized via env.reset().
2. At each turn:
    - A challenge decision is made.
    - If challenge is successful or skipped, a move decision follows.
    - Each action triggers a state transition and returns a reward.
3. Replay memory is updated (if in training).
4. Target networks are updated periodically (if in training).
5. Current game's action selection and success counts are collected (if LLM agent is playing)

6. Game ends when a player empties their hand or maximum turn limit is reached. When a player empties their hand before the game reaches turn limit, the player wins the game. Otherwise, there is no winner for that game.

Throughout the episode, each DQN agent's internal counters, memory length, and epsilon values are logged at periodic intervals for training inspection.

4.3 Gameplay Constraints

In experiments played with real people, we observed that the fixed table rank feature in the game makes the opponents' cards predictable when the number of rounds is extended, and if the game goes on for too long, the game can enter an endless loop. We assumed this could be a situation we might encounter, especially with learning-based RL agents. Therefore, to make the game environment cost-effective for LLM agent and to enable learning-based agents to play more effectively, we set a maximum number of turns that we control for each game. This turn limit is intended to ensure that the game ends before the cards in the players' hands become too predictable in a three-player environment. We chose 100 turns as the maximum number of turns. We canceled games that did not end after the 100th turn and set a rule that no winner would be determined in that game. We aimed to reduce the number of turns by adding this negative feedback, or rather a penalty, to the reward methods of RL agents.

4.3.1 Training set up

AI agents, while learning to play the game, expose themselves to diverse situations and experiment with a variety of actions required for each state during their learning process.

Training is performed on mini-batches of size 64, sampled uniformly from $\mathscr{D}$. This approach decouples data collection from learning and improves stability by reducing correlations between consecutive experiences.

Agents are trained for 2000 episodes, with each episode representing a full game involving three players. A turn limit of 100 was enforced; games that exceeded this

limit were terminated, and large penalties were assigned to involved agents and the game is end without a winner. In addition to that, in each episode:

- The game is initialized with a random player order, random card distribution.
- Agents interact with the environment via env.step() calls.
- The DQN agent updates its policy for each action in challenge phase since outcome of a move occurs at next turn, both the move reward of a current player and next player's challenge action are determined based the next player's challenge action outcome. Moreover, the policy network synchronizes with the target network every 20 environment steps which is every 5 games in average, which corresponds to either a challenge or move action. This ensures stable Q-value updates.
- Rewards are logged separately for move and challenge actions.

The state observations used by the DQN are derived from public game variables (e.g., hand sizes, pile stats), excluding private card information to reflect partial observability.

4.3.1.1 Training configurations of RL agents

Four DQN agents were trained under two different configurations to compare learning-based strategies: (i) a baseline-oriented setup which is described as a matchup configuration where the agent is trained against rule-based opponents with fixed behaviors, and (ii) a self-play setup which is described as where agents learn by competing against identical DQN agents, allowing them to adapt to evolving strategies and generalize through varied in-game interactions. Although agents share the same training setup and opponent pool in self-play settings, they follow independent learning trajectories and thus yield distinct model weights because of scholastic elements of the game. For this purpose, we create four distinct DQN-based RL agents as below

- **RL_Agent_0**: Trained against Agents with Bayesian and State-Dependent strategy in baseline-oriented simulation framework.
- **RL_Agent_1**, **RL_Agent_2**, **RL_Agent_3**: Trained using self-play under same reward shaping method.

This setup enables the evaluation of the DQN agent in both fixed and dynamic environments. RL_Agent_0 learns in a stable setting with predictable opponents, allowing it to optimize against known behaviors. In contrast, RL_Agent_1–3 uses self-play to adapt to evolving strategies since they all learn the game and optimize their actions progressively during the training.

All RL agents follow an ε-greedy policy [25], where ε starts at 1.0 and decays gradually per episode to 0.01. Epsilon decay is applied at the end of each episode by multiplying the exploration rate with the decay factor $\varepsilon \leftarrow \varepsilon \cdot \varepsilon_{\text{decay}}$, until it reaches the minimum threshold ε_{min}. At each step, the agent either selects a random action (exploration) or follows the policy network (exploitation) which depends on the current epsilon value. During playing and at the end of each episode, action rewards and final rewards are assigned depending on the win/loss outcome and stored into the replay buffer, respectively. End game transitions are marked with a done flag, which ensures the agent correctly learns both from intermediate gameplay and from the terminal condition that ends the game.

In figure 4.2, we explain how the decision-making process works for DQN agents. The agent makes two sequential decisions per turn: a *challenge action* and a *move action*.Each action receives its own reward and initiates a distinct state transition, enabling independent learning updates. This structure captures the dual-decision dynamics of the game and uses targeted Q-value learning via the Bellman equation.

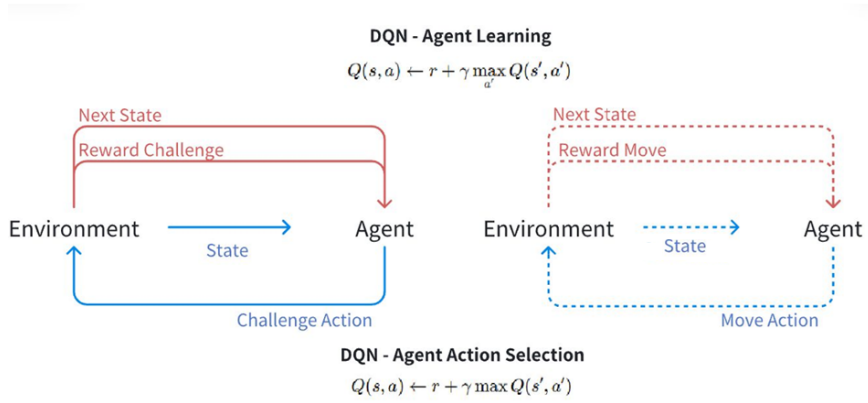

Figure 4.2: DQN Agent's Decision Making Flow: Selects actions using learned Q-values based on state and reward feedback.

4.3.1.2 Reward functions for baseline-oriented and self-play training

The reward function is designed to guide the agent towards optimal decision-making by providing distinct feedback for challenge and move actions, which can be described as turn-based reward. In addition, since the game consists of multiple turns and is concluded by discarding all cards from a hand, we also define end game reward to highlight the win condition by learning the terminal condition. Also, to reduce the number of turns, we introduce a turn factor that shapes the end game penalty and move rewards. In this way, agents learn to finish the game as early as possible and avoid reaching the turn limit. So, reward values are shaped separately for the outcome of each action option and include additional contextual feedback processed outside the environment step as post-decision evaluations such as turn-based penalties, endgame win/loss assessments, and opponent-dependent conditions (e.g., rewarding successful challenges when the opponent has no cards left, rewarding successful bluffs when the opponent does not challenge it).

In the baseline-oriented training setup, moderate rewards and scaled penalties are applied as in Table 4.4.

Table 4.4: Reinforcement rewards in the bluff-based card game based on agent actions and outcomes for baseline-oriented training configuration.

Action / Condition	**Reward**
Challenge Action Rewards	
Challenge succeeds (opponent has 0 cards)	$+5$
Challenge succeeds (opponent has > 0 cards)	$+3$
Challenge fails	-2
No challenge	0
Move Action Rewards (based on opponent's response)	
Truthful move is challenged	$+3$
Bluff succeeds (not challenged)	$+3$
Truthful move unchallenged	$+3$
Bluff fails (challenged)	-3.5
No move	0
Game Outcome Rewards (terminal reward)	
Win the game (turn $>$ 10)	$+15 - 10 \times \gamma_t$
Win the game (turn $\leq$ 10)	$+15$
Lose the game	-10
Game reaches max turn	-15

where γ_t is the turn factor dependent on the next state's turn number as below:

$$\gamma_t = \frac{t+1}{(\text{Turn\_UL} = 100)} \tag{4.27}$$

which is the maximum limit of the number of turns as 100. This factor allows us to reduce the number of turns and encourages performing bluff actions in the early game while helping to avoid noisy learning caused by ineffective late decisions. So, failures are penalized proportionally to the turn factor to discourage inefficient late-game behavior. This setting is designed for stable learning against predictable opponents.

With this approach, we demonstrate the importance of the challenge action by giving more reward compared to the penalty for failure. This is because all agents have to bluff at some point, so to win the game, the DQN agent must learn when to bluff. To highlight the importance of challenging when an opponent agent has no cards left, we give extra reward for successfully challenging and catching a bluff. In such cases, the agent sacrifices ending the game and becomes the loser, which makes the correct challenge even more valuable.

However, when used in self-play, this reward scheme results in longer episodes and overly aggressive behaviors for all self-play agents, which result in high bluff and challenge frequencies. To address this, we define a second reward function as in Table 4.5 for self-play training.

This configuration introduces a fixed penalty for failed moves which is greater than successful actions to indicate the risk of bluff and reduce failed bluff actions by discouraging inefficient or overly risky play. It also uses a fixed penalty for failed challenges for clarity of action selection and reduces variance in learning signals. It offers high rewards for successful truthful actions to promote effective and stable strategies. Unlike the first configuration, this version applies more consistent and stronger penalties regardless of the game phase, which helps reduce episode length and stabilize agent behavior.

Table 4.5: Reinforcement rewards in the bluff-based card game based on agent actions and outcomes for self-play training configuration.

Action / Condition	Reward
Challenge Action Rewards	
Challenge succeeds (opponent has 0 cards)	$+5$
Challenge succeeds (opponent has > 0 cards)	$+3$
Challenge fails	-2
No challenge	0
Move Action Rewards (based on opponent's response)	
Truthful move is challenged	$+5$
Bluff succeeds (not challenged)	$+3$
Truthful move unchallenged	$+3$
Bluff fails (challenged)	-3.5
No move	0
Game Outcome Rewards (terminal reward)	
Win the game (turn $>$ 10)	$+15 - 10 \times \gamma_t$
Win the game (turn $\leq$ 10)	$+15$
Lose the game	-10
Game reaches max turn	-15

4.4 Opponent Filtering Method for LLM agent Evaluation

To evaluate the performance of the LLM-based agent in a structured, meaningful, and cost-efficient way, we designed a multi-stage elimination method to narrow down the number of agents it would compete against. The primary goal was to reduce the number of match-ups by eliminating underperforming agents prior to evaluation, while preserving agent diversity and strategic competitiveness.

As a first step, we focused on the baseline agents. These include agents using probabilistic or statistical reasoning which are the Bayes and State-Dependent agents, alongside a Random agent used as a benchmark. We conducted internal tests where baseline agents played against each other, and based on aggregated performance metrics, we eliminated agents that failed to significantly outperform the random strategy. Agents with consistent advantage were added to the list of potential LLM-based agent's opponents. These selected agents serve as references for both training and evaluating the RL agent, as well as assessing the action performance of the LLM agent.

In the second stage, we focused on the performance of reinforcement learning agents. We trained four DQN agents under two different configurations: one trained in a *baseline-oriented* environment (against rule-based agents) and the other three through *self-play*. After training, the baseline-oriented agent and the top two self-play agents were selected based on their internal training performance. These three agents were then tested against one another in the game environment. According to the simulation result, the two DQN agents that achieved the best overall performance in terms of performance metrics are selected and added to the candidate pool.

As a result of this elimination process, we reduced the original set of 8 agents to a competitive group of 5. This reduction allowed us to decrease the size of the combination space from $\binom{8}{3} = 56$ unique match-ups to only 10.

Among these 10 match-ups, 6 included the LLM-based agent. The remaining 4 consisted solely of RL and baseline agents. These 4 games were used to identify the strongest baseline and strongest RL agent. Based on the results that we obtained from tournament-style elimination, we further narrowed down the LLM agent evaluation scope from 6 to 3 match-ups by selecting the most meaningful cases:

- LLM agent vs. two baseline agents,
- LLM agent vs. two RL agents,
- LLM agent vs. the strongest baseline and strongest RL agent.

This final evaluation setup provides that the LLM agent is tested only against high-performing opponents and same type of agents (RL or baseline), eliminating uninformative comparisons. It also allows us to understand the LLM agent's behavioral pattern under different game setups where making inference and prediction is important for successful deception and measures whether the LLM agent can match or outperform agents that have been optimized through domain-specific training in terms of bluff move and challenge selection, thereby providing a focused and fair comparison across strategy types.

In conclusion, by using this agent filtering method, a subset of agents that shows strong performance in preliminary tests can be chosen for LLM-agent performance evaluation under deceptive nature of bluff card game, which answers our research question.

Corresponding match combinations of these agents are constructed by using a predefined configuration file (`strategy_combinations.json`). This modular setup ensures consistency and reproducibility across simulations for all 3-player matchups.

The agent match ups for all steps of elimination (filtering) and final evaluation set includes:

- One test configuration where all three agents follow baseline strategies,
- One configuration between top three DQN-based RL agents (used for testing, not training),
- Four configurations where baseline and RL agents are mixed to evaluate cross-type competition,
- Three configurations involving the LLM-based agent.

In total, nine distinct evaluation configurations were simulated. This setup ensures that the LLM agent is only compared against strong-performing agents, and that each agent type is tested both in homogeneous (same-type) and heterogeneous (mixed-type) match-ups.

4.5 Performance Metrics and Evaluation

To evaluate and compare agent behaviors in the bluff-based card game, we recorded detailed gameplay outcomes for each agent based on opponent filtering method and number of episodes that they play (2000 for training and baseline validation, 1000 for filtering, 250 for LLM evaluation). These simulations do not involve learning during testing phases; instead, they are performed with fixed pretrained agents using their final model weights, thereby reflecting the outcome of their prior training phase under baseline-oriented or self-play setups.

We collect game and player-specific variables that are related to actions and their outcomes to analyze behavioral dynamics as listed below:

1. Total number of games played
2. Number of games that reached maximum turn limit (100 turns)
3. Total number of turns played
4. Total number of challenge/move opportunities, executed challenge/move/bluff move actions, and mandatory bluff cases (when no matching cards were available)
5. Successful bluff and challenge counts
6. Average number of cards played per move

These variables represent how each agent performs in the overall simulation.

Moreover, the following performance metrics are examined for the evaluation of agents' performance and are updated at the end of each episode/game:

1. Cumulative win count: Total number of episodes won by each agent.
2. Average win rate of agents are smoothed using a rolling window as every 20 episodes to analyze behavioral stability.
3. Bluff selection rate: The proportion of moves declared as bluff.
4. Bluff success rate: The frequency of bluff moves that were not challenged and resulted as in advantage of the agent.
5. Challenge selection rate: The proportion of selecting the challenge action among all available challenge turns.
6. Challenge success rate: The percentage of challenges that correctly identified a bluff.
7. Turn distribution: The number of turns performed to conclude each episode, used to estimate strategic efficiency.

8. For training phases of DQN agents following metrics also examined:

    (a) Move and Challenge Rewards: Cumulative move and challenge rewards of action selection across simulation.

The evaluation framework is designed to be reusable across simulation setups where agent configurations vary. At the end of each game, all game data, which contains player actions, observed states, outcomes, and rewards, are saved into structured JSON logs by a RecordGameplay module. Metrics are computed per agent using the aggregated logs and visualized using rolling averages to observe long-term behavior stability. In addition to cumulative win counts, challenge/move rewards, and success rates, confidence intervals are applied to probability curves such as bluff and challenge success over time to account for stochastic variance.

These indicators help to expose deeper behavioral dynamics, such as whether an agent is overly aggressive, risk-averse, or balanced and performs effective deceptions. The primary purpose of evaluation is twofold: (1) to assess end-game success, such as win rates and episode efficiency, and (2) to analyze action-level decision effectiveness, such as bluff and challenge accuracy, and their behavioral tendencies across games. Thus, action-level and game-level statistics are aggregated at the end of simulations to create performance curves and summary tables. In this way, we are able to gain insights into both the consistency and adaptability of agents and their comparative effectiveness in strategic bluff-based gameplay.

5. ANALYSIS

In this section, we evaluate and compare the effectiveness of various agent decision-making systems by analyzing their actions, success rates, and overall performance outcomes. For the evaluation, we analyze each matchup by simulating them under the same game environment.

First, we compare our designed baseline strategies by assigning them to agents. The agent using random decision-making is considered as the benchmark baseline strategy. Then, we evaluate the agents' decision-making behaviors, their action success rates, and overall win rates to validate the effectiveness of the State-Dependent and Bayesian strategies.

Next, we train reinforcement learning (RL) agents in two different configurations. Based on performance metrics, we initially select the top three agents among the four candidates. After that, we evaluate the performance of these three agents and select the top two RL agents according to their results. As part of an elimination and opponent filtering methodology to select opponents of LLM agent, we conduct four match-ups combining three agents, a combination of three of the top four (2 RL and 2 baseline agents). Following these match-ups, we determine the best-performing baseline and RL agents.

Finally, we design three match-ups to evaluate the performance of the LLM agent. These include:

1. LLM agent vs. learning-based agents,
2. LLM agent vs. baseline agents,
3. LLM agent vs. best-performing agents among baseline and learning-based agents.

According to the simulation results, we evaluate action frequencies to understand the agents' risk-taking tendencies, and action success rates to assess their decision-making

performance. These metrics provide insights into the agents' effectiveness in opponent modeling and their perception under deception. Additionally, we examine whether there is a correlation between action success and winning, especially under the influence of deceptive scenarios and stochastic elements player order. Finally, win rates are analyzed to determine overall performance efficiency. This allows us to compare different decision-making systems and identify the most effective one. These systems include decision-making mechanisms based on purely random selection, short-term probabilistic reasoning using current state information, statistical inference from limited observations, and either reinforcement learning or in-context reasoning based on pre-trained knowledge.

To assess the reasoning capacity of the LLM agent, we compare its results against all other agents. This evaluation aims to answer the question: Can an LLM provide effective guidance in bluff-based, deceptive game scenarios? Furthermore, we investigate the LLM's behavioral tendencies, its ability to infer opponents' strategies, and its success in various bluff-related decision points.

5.1 Simulation Results

In this section, we represent the analysis of the simulation results and evaluate the performance of each agent in different match-ups.

5.1.1 Performance metrics and evaluation of baseline agents

This section presents the outcomes of baseline agents: Random, State-Dependent, and Bayesian by evaluating their action frequencies and strategic effectiveness. In this way, we are able to determine the performance of designed baseline agents by comparing them with the benchmark baseline agent.

Baseline agents simulation sample size and sample that reach the max turn limit are given in Table 5.1.Furthermore, based on the results provided in Table 5.2, we can notice the behavioral differences between baseline strategies. Since each of them has a different decision-making process and different memory configurations, the random strategy has no memory, the state-dependent strategy has only the current turn

observation, and the Bayesian strategy has both current turn observation and past game statistics.

Table 5.1: Total number of games simulated and number of games that reached the turn limit in baseline match-up

Metric	Value
Games Played	1000
Games (100 Turns)	1

Table 5.2: Performance metrics of agents with baseline strategies.

Metric	Bayes	Random	State-Dpd
Total Turns	7825	8009	7919
Move Turns	6990	6487	7295
Total Bluffs	3717	3730	3847
Forced Bluffs	242	285	277
Successful Bluffs	1793	1895	1768
Challenge Turns	6491	6915	6366
Total Challenges	3135	3442	2472
Successful Challenges	2211	1870	1757
Avg Cards / Move	1.340	1.315	1.314

The Random agent issued the highest number of challenges despite having a lower success rate as 54.3% compared to the baseline agents. Bayes agent succeeded in 70.5% of its challenge action. This suggests that the Bayesian agent is more selective and effective in identifying deception, leveraging probabilistic inference over random choice.

In terms of bluff behavior, the State-Dependent agent made the highest number of bluffs as 3847, with a success rate of 45.9%. Interestingly, the Random agent achieved the highest bluff success rate (50.8%), even though its bluff count was lower and similar to the Bayes agent. This may be because of the chance-based factor of the game. The Bayesian agent, on the other hand, maintained a balanced bluff profile with a moderate count of 3717 and a success rate of 48.2%, implying a more calculated bluff strategy and adapting to its environment.

The number of forced bluffs, which can be defined as situations where agents had no matching cards, was highest for the Random agent, which indicates its inability to optimize hand management when we compare the baseline agents.

In contrast, the Bayesian and State-Dependent agents had fewer forced bluffs, suggesting better control over card play and more deliberate risk exposure, which is a result of matched card evaluation.

Moreover, in terms of card playing, all agents displayed similar average cards per move values, with the Bayesian agent playing slightly more aggressively by playing 1.340 cards per move, hinting at its confidence in calculated risks. The State-Dependent agent, although cautious in challenges, had the highest number of move turns, which is 7295, suggesting more frequent opportunities for safe play, possibly due to its reliance on the current game state for decision-making and low rate of challenge action on opponents.

The results highlight meaningful behavioral differences between the three baseline strategies. While all agents participated in the same number of games, their decision-making frameworks led to distinct patterns in action selections. In parallel to memory capacity, the simulation resulted in the best-performing agent having the highest win rates.

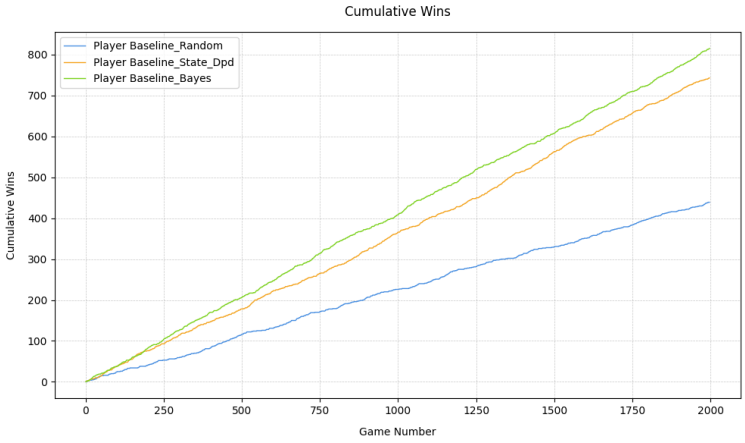

Figure 5.1: Win Rates of Baseline Match-up

In addition to that, these metrics indicate that while the Random agent exhibits high activity levels, its lack of strategic insight results in lower success consistency. Additionally, the Random agent's bluff frequency reflects the influence of mandatory bluff scenarios by performing bluffs more than %50 percent. The Bayesian agent demonstrates superior efficiency when we compare the others in both challenge and bluff execution, owing to its probabilistic reasoning, which leads it to become the most

successful agent among these 3 players. Meanwhile, the State-Dependent agent adapts to observable conditions, maintaining a balanced gameplay experience and shows the effectiveness of state by using it to evaluate the probability of cards with table color in opponents' hands.

In conclusion, by using the random agent benchmark, we validate the decision-making effectiveness of State dependent and Bayes agents. These agents serve as fixed behavioral references for comparison with learning-based and reasoning-based agents.

5.1.2 Performance metrics and evaluation of learning-based agents in training phase

This section examines the training performance of DQN-based reinforcement learning agents. We explore their cumulative wins, average wins per 20 episodes, frequencies, rewards, and success of selection of deceptive actions both challenge and bluff, and stability throughout the training phase under both baseline-oriented and self-play setups.

5.1.2.1 Baseline-oriented RL agent training result

All three agents played 2000 games in a unified game environment, and 56 of those games reached the 100 turn cap, which, as a result of that, had no winner in 56 games. The result of the simulation for each agent shown in Table Table 5.3 highlights several important differences in how the agents behave and perform in the overall sample. In contrast to the match between baseline agents, game sessions between the best baseline agents and the RL agent are longer. This may be because of the learning phase of the RL agent and the inference ability about the opponents' cards of the RL agent.

Table 5.3: Games in baseline-oriented training.

Metric	Value
Games Played	2000
Games (100 Turns)	56

In Table 5.4, frequency of action selections for each agent are shown which is used to identify behavioral differences of agents.

Table 5.4: Performance metrics comparison: RL_Agent_0 vs. baseline agents.

Metric	RL_Agent_0	Bayes	State-Dpd
Total Turns	17570	17373	18103
Move Turns	16690	14870	16926
Total Bluffs	12955	8824	9419
Forced Bluffs	1134	202	141
Successful Bluffs	5273	3600	2155
Challenge Turns	15202	15798	15490
Total Challenges	8121	11098	6139
Successful Challenges	7231	8503	4388
Avg Cards / Move	1.323	1.360	1.483

Based on the result table, RL_Agent_0 shows a clear tendency toward high activity, especially in terms of bluffing. It performed 12955 bluffs which is much more than Baseline_Bayes which made only 8824 bluff moves and Baseline_State_Dpd which performed 9419 bluff moves. It also had the highest number of mandatory bluffs, which occur when an agent has no matching cards and is forced to bluff.

According to this result, we can observe that RL_Agent_0 frequently ended up in high-risk situations, possibly due to its exploratory behavior during training and also learning about opponents' challenge patterns based on reward.

Despite this aggressive bluffing pattern, RL_Agent_0 managed to succeed in 5273 bluff attempts. While this is the highest number of successful bluffs overall, it shows the bluff success rate around 40.7% which is slightly lower than that of Baseline_Bayes with 40.8% bluff success rate. On the other hand, Baseline_State_Dpd shows the lowest success rate, which had only 22.9% successes. This implies that RL_Agent_0 exhibited a higher tendency to bluff, and the interaction between reward feedback and state dynamics has supported successful learning, given that it achieved the highest number of successful bluffs when we compare other agents' success. Thus Baseline_State_Dpd shows a non-effective strategy by using only state variables since the RL agent understands the situation-specific behaviors with its reward shaping model, and Baseline_Bayes follows its statistical learning by combining state information and adapting general behavior of both agents. Cumulative move rewards across all turns in all games based on reward table 4.4 for RL_Agent_0 shown in Figure 5.2.

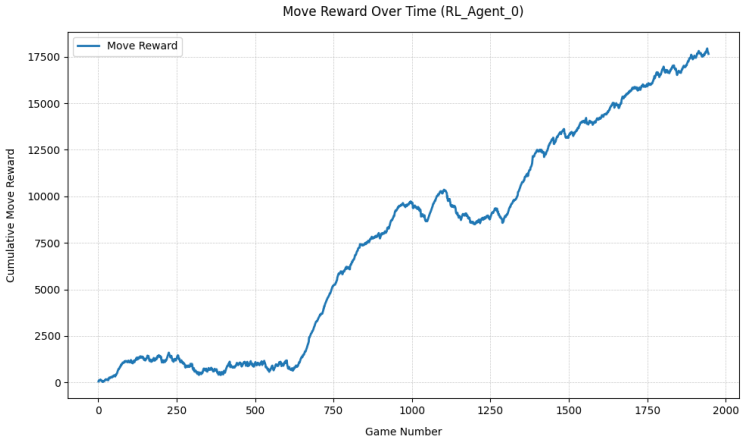

Figure 5.2: RL_Agent_0's cumulative move reward

For the challenge action frequencies, Baseline_Bayes made the most attempts, which is 11098, and also succeeded the most as 8503 of them resulted successfully, giving it a success rate of roughly 76.6%, which reflects its strength in probabilistic inference. Moreover, RL_Agent_0 shows a higher success rate for challenge action while the number of challenges is less than Bayesian, as 7231 were successful out of 8121 challenge attempts, which resulted in a success rate of 89.0%, which is the highest among the three. This suggests that the agent has learned to identify bluffs more accurately over time by using reward feedback and state dynamics, which makes it possible for the DQN agent to observe patterns in opponents, which are given in Table 4.4 for RL_Agent_0 shown in Figure 5.3.

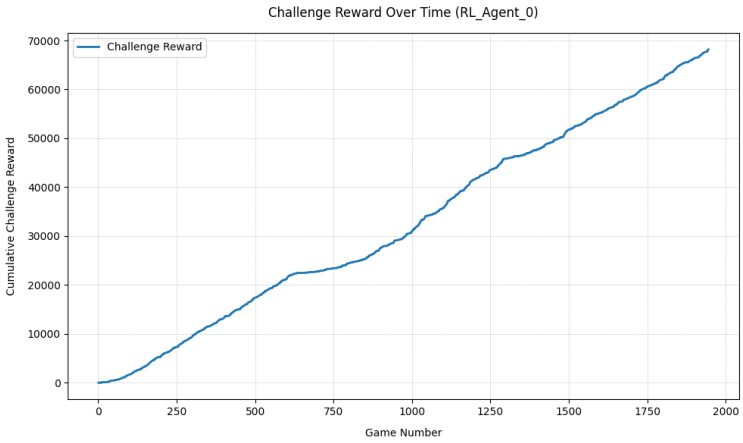

Figure 5.3: RL_Agent_0's cumulative challenge reward

In terms of average cards played per move, Baseline_State_Dpd played the most aggressively , followed by Bayes, and then RL_Agent_0 by playing 1.483 cards, 1.360 and 1.323 cards respectively. The lower value for RL_Agent_0 may indicate that it chooses to conserve cards or minimize exposure when the risk of being challenged is high based on its learning method.

As a result of these action decision-making processes, we show that the DQN-based RL-Agent combines a high bluff frequency with strong challenge accuracy. Compared to the top two baseline agents, RL_Agent_0 stands out as a learning-based agent that can adjust its decisions based on accumulated experience, making it particularly well-suited for complex, deceptive environments.

RL_Agent_0 shows rapid early improvement as its mean win value increases as the number of games increases for the first 250 games and reaches a stable win rate above 0.45 after that, which is shown in figure 5.4. Its performance remains consistent with minimal fluctuations, which can be related to chance-based parameters in initial conditions.

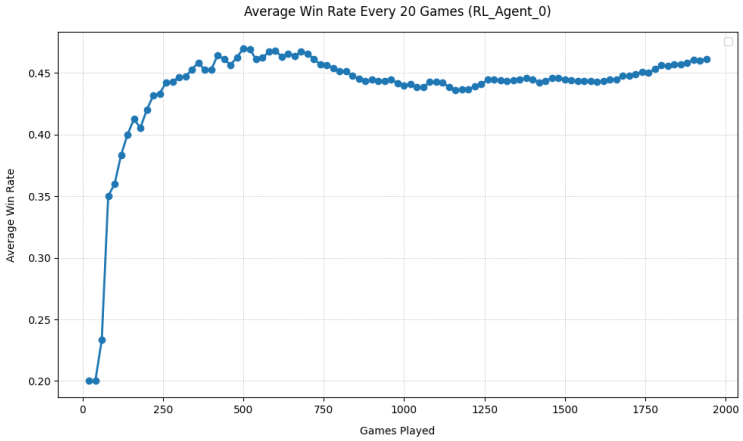

Figure 5.4: Average win per 20 levels for RL_Agent_0

Furthermore, the cumulative win graph in figure 5.5 confirms its dominance by showing consistently outperforming both Bayesian and State-Dependent baseline agents. Thus, we can say that RL_Agent_0 demonstrates efficient learning and sustained strategic success across all episodes for this training match-up.

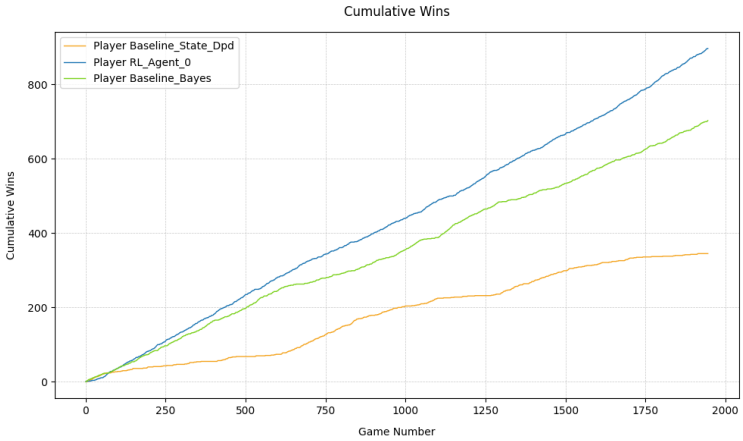

Figure 5.5: Win Rates of Baseline-Oriented Training

5.1.2.2 Self-play training result

After we test the DQN-based agent with baseline agents, we set up the second training configuration, which is self-play training. The self-play training results, as presented in Tables 5.5 and 5.6 show clear differences in the strategic behavior and learning outcomes of RL_Agent_1, RL_Agent_2, and RL_Agent_3 when we compare the training results of the baseline-oriented RL agent. All agents completed 2000 games, and 61 of them reached the 100-turn cap, meaning they were exposed to similar long-term decision-making situations, which is expected due to the self-play setting.

Table 5.5: Games in self-play training.

Metric	Value
Games Played	2000
Games (100 Turns)	61

RL_Agent_3 shows the most balanced and effective behavior for bluff action which is shown in the cumulative move reward graph in Figure 5.6. It made the highest number of bluffs as 8198 and had the most successful bluff count as 2020, with a 24.6% success rate. At the same time, it had the lowest number of forced bluffs as 85, indicating that it bluffed voluntarily rather than out of necessity and succeeded in some of them in an aggressive game environment, so we can say that the agent may save its truthful cards during the early stages of the game.

Table 5.6: Performance metrics of self-play RL agents.

Metric	RL_Agent_1	RL_Agent_2	RL_Agent_3
Total Turns	13058	12957	12973
Move Turns	10916	10743	11306
Total Bluffs	7780	7199	8198
Forced Bluffs	161	160	85
Successful Bluffs	1225	1227	2020
Challenge Turns	10317	10444	10218
Total Challenges	7911	8914	7898
Successful Challenges	5762	6691	6218
Avg Cards / Move	2.434	2.542	2.632

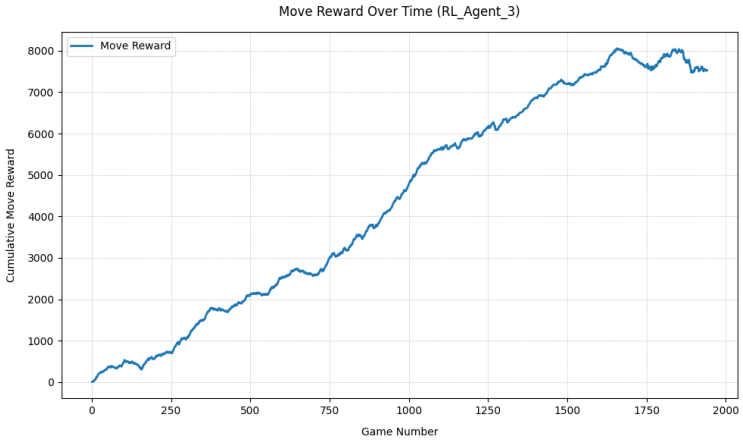

Figure 5.6: RL_Agent_3's cumulative move reward

In terms of challenges, RL_Agent_3 made 7898 attempts and succeeded in 6218 of them, reaching a 78.7% success rate, which is the highest among the three agents as seen in the cumulative challenge reward graph which is shown in Figure 5.7.

This shows that RL_Agent_3 supports its bluff strategy with well-learned challenge action and detects opponents' bluffs effectively compared to others. Also, its average of 2.632 cards per move suggests that it played more assertively when it had an advantage.

RL_Agent_2 leads in challenge attempts at 8914 and has a high success count as 6691, corresponding to a 75.1% success rate. Its bluff success rate is around 17.0%. For the RL_Agent_1 gameplay results, we can say that it performs more cautiously. It made fewer bluffs at 7780 and had the lowest bluff success count of 1225, which resulted in a 15.7% success rate. It also had the lowest challenge success rate at 72.9%. This suggests that the policy it learned was more passive or it affected the initial conditions

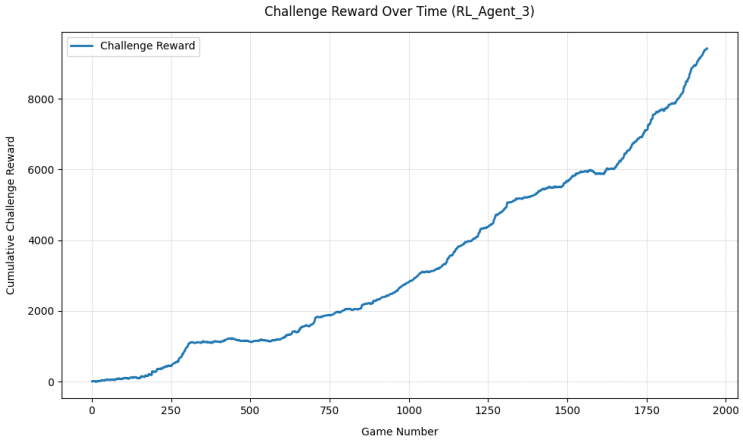

Figure 5.7: RL_Agent_3's cumulative challenge reward

of games more than others. This shows that even if RL_Agent_1 and RL_Agent_2 can detect bluffs well, they are not as efficient in bluffing compared to RL_Agent_3, but high challenge success can occur because of the high bluff rates of the agents.

When we compare win rate trends in Table 5.8, RL_Agent_3 again stands out. Its win rate steadily increases and surpasses 47% with minimal fluctuation after the 1000th game, as given in Table 5.9. Other agents show earlier performance drops or inconsistencies. Overall, RL_Agent_3 appears to be the most stable and generalizable agent in self-play.

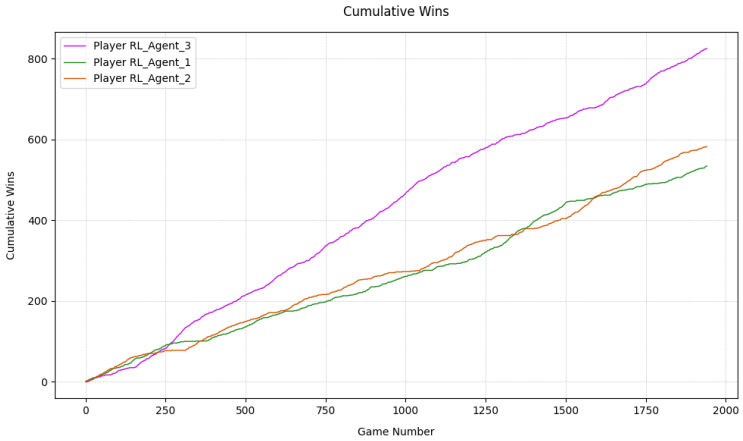

Figure 5.8: Win Rates of Self-play Training

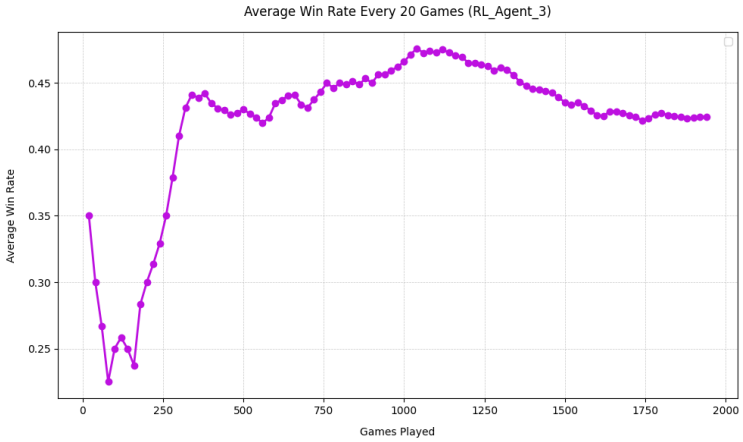

Figure 5.9: Average win per 20 levels for RL_Agent_3

5.1.3 Performance metrics and evaluation of trained learning-based agents

This section evaluates how trained RL agents behave during a simulation with no further learning. The goal is to assess their strategic consistency, action preferences, and overall effectiveness using the policies obtained after training. (top two self-play and a baseline-oriented trained agents) that we mentioned 5.1.2.

After the training phase finished and the top three RL agents are determined, we compare the overall behavioral trends and effectiveness of RL_Agent_0, RL_Agent_1, and RL_Agent_3 during the 1000-game evaluation phase based on the quantitative metrics of valid games and invalid games in Table 5.7.

Table 5.7: Games between trained RL agents.

Metric	Value
Games Played	1000
Games (100 Turns)	30

The table 5.8 shows that all agents have high bluff frequency when they play with each other. RL_Agent_0 consistently receives high bluff probability estimations approximately 90%, yet its actual bluff success rate remains the lowest at around 13%, indicating predictable or ineffective bluff timing. RL_Agent_3 achieves the highest bluff success rate as 30% with moderate bluff frequency as 70%, suggesting

better timing and risk management. RL_Agent_1 falls in between, with bluff success stabilizing near 24%.

Table 5.8: Evaluation performance metrics of RL agents (1000-game test).

Metric	RL_Agent_0	RL_Agent_1	RL_Agent_3
Total Turns	8216	8350	8281
Move Turns	7607	6809	7260
Total Bluffs	6424	5162	4924
Forced Bluffs	17	44	101
Successful Bluffs	806	1333	1616
Challenge Turns	6163	7396	7130
Total Challenges	3254	6370	6298
Successful Challenges	2645	4829	5277
Avg Cards / Move	1.821	2.827	2.990

However, their challenge frequency reflects clear differences. While RL_Agent_3 and RL_Agent_1 challenge their opponent's move with high frequency, RL_Agent_0 shows a moderate challenge frequency since it learned the game in more conservative settings.

The result also shows that RL_Agent_3 has the highest success rates in bluff and challenge success as 30% and 86% respectively. RL_Agent_1 followed with approximately 24% bluff success and 80% challenge success. Lastly, RL_Agent_0 had the lowest bluff success and challenge success as 15% and 76% respectively. These values suggest that RL_Agent_3 executes bluffs and detects deception more effectively than the others. These success rates indicate when every agent in gameplay is aggressive, which means every agent in the game has high challenge declaration rates, the chance of getting a successful result from a bluff is very difficult.

Compared to the self-play training phase, RL_Agent_3 and RL_Agent_1 maintained their performance consistency during evaluation, showing similar success rates in both bluffs and challenges. However, RL_Agent_0 showed a decline in both actions' effectiveness compared to training. These results also explain how self-play agents learn to select actions in a more generalized environment than RL_Agent_0 with baseline-oriented training.

Moreover, average win graphics of these agents are given in figures 5.10-5.12. RL_Agent_0 shows a sharp decline in win rate early on, stabilizing around 14.5%.

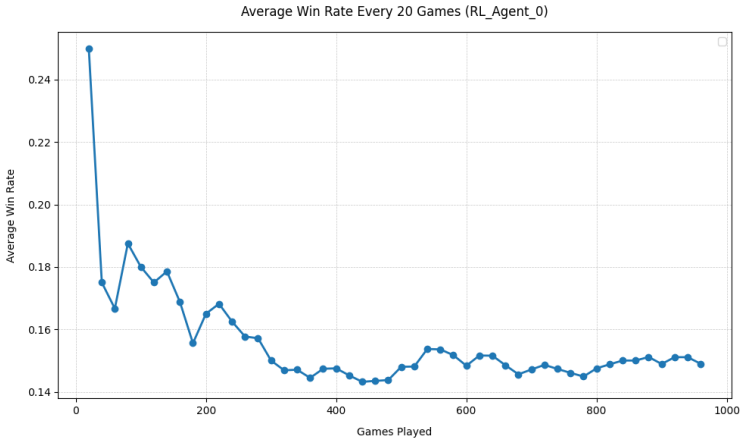

Figure 5.10: Average win per 20 levels for RL_Agent_0

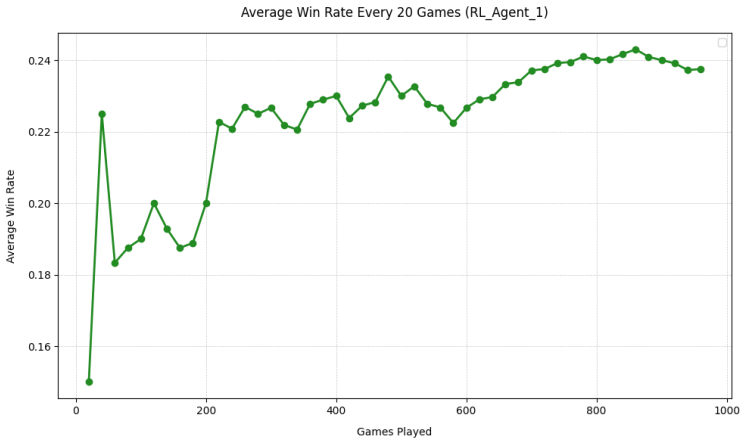

Figure 5.11: Average win per 20 levels for RL_Agent_1

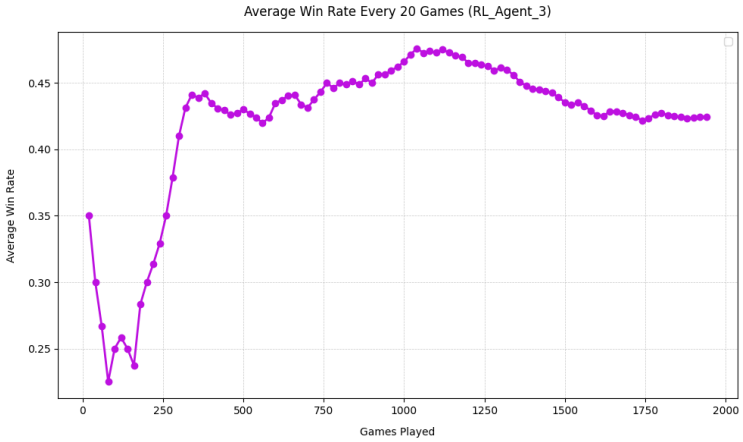

Figure 5.12: Average win per 20 levels for RL_Agent_3

In contrast, RL_Agent_3 displays a clear upward trend in win rate and finishes with the highest cumulative wins, indicating strong and consistent performance.

For the opponent filtering phase, we select two agents as RL_Agent_3 and RL_Agent_0. Although RL_Agent_1 demonstrated a better win performance than RL_Agent_0, we decided that due to differences in learning, nominating RL_Agent_0 for the next stage was a more logical choice for comparing different agents. Because self-play agents tend to have a more aggressive and aggressive play style based on the average number of cards played and their challenge frequencies, this translates to a significantly different gaming experience than RL_Agent_0's training environment. For a better comparison, we selected RL agents with two different learning configurations as opponents in the next stage, which is the performance evaluation of LLM agent with different match-ups.

5.1.4 Performance metrics and evaluation of match-ups in the candidate pool of LLM agent's opponents

In this stage, we compare RL_Agent_3, RL_Agent_0, Baseline_Bayes and Baseline_State_Dpd in different match-up combinations which are

1. RL_Agent_3 - RL_Agent_0 - Baseline_Bayes,
2. RL_Agent_3 - RL_Agent_0 - Baseline_State_Dpd,
3. RL_Agent_3 - Baseline_State_Dpd - Baseline_Bayes,
4. RL_Agent_0 - Baseline_State_Dpd - Baseline_Bayes.

According to the results that we obtained, Baseline_Bayes is the most successful baseline agent in terms of execution of successful bluff and challenge, and also win rates. Also, RL_Agent_0 becomes the most successful agent in these match-up sets as a learning-based agent based on evaluation metrics. For a similar reason to the RL agent matchup, we observed no difference in training configurations between baseline and RL matchups. The lower performance of the self-play RL agent compared to the baseline-oriented RL agent against baselines may be due to its risky play style emphasizing the high probability of bluffing in card counting strategies. Furthermore,

we observed that the bluffing frequency of the games played directly influenced the Bayesian agent's challenge strategy, leading the self-play agent to outperform the Bayesian agent.

5.1.5 Performance metrics and evaluation of match-ups for LLM agent

To evaluate LLM_Agent's reasoning-based performance, we conducted a 250-game evaluation of three different player matchups in the same game environment. In this section, we described how the LLM agent reasoned in different player setups against opponents with different strategies. We examined the LLM agent's frequency of deceptive action selections, the frequency of catching opponents' bluffs and the success rate of these actions, and the frequency of game wins. We also examined whether there was a correlation between wins and turn-based performance.

5.1.5.1 LLM interpretability via in context learning

To increase transparency and interpretability, the LLM agent is prompted to return not only the selected action, but also a short natural language explanation. The agent receives structured prompts including the current state, current game memory, and prior statistical patterns of opponents. In addition, we also provide strategy guidelines about how to infer opponents' behavior, and how to use this information to make optimal actions. It then returns its decision alongside a justification reflecting its reasoning process.

These decisions, provided for both challenge and move phases, enable a richer evaluation of model behavior. They are particularly useful for identifying strategy patterns, reasoning errors, or potential biases. Sample outputs for challenge and move actions are shown below, respectively:

- "challenge": 0, "justification": "The opponent played only 1 card, and the pile is small with just 1 card. The probability of a bluff is not high enough to justify the risk of challenging, especially given the small pile size and the fact that a failed challenge would put me in a worse position. Additionally, the opponent has 3 cards left, which is not an immediate threat to win. Therefore, passing is the safer option."

- "move_type": 1, "amount": 2, "justification": "I have 1 matched card and 7 unmatched cards. The pile is small with only 1 card, making it a low-risk opportunity to bluff. By bluffing with 2 unmatched cards, I reduce my hand size while preserving my matched card for future turns. The next player holds 12 cards, increasing the likelihood they may not challenge due to their own need to reduce hand size. This move maintains flexibility and unpredictability for future turns."

5.1.5.2 Learning-based and reasoning-based agent match-up for LLM agent's performance evaluation

In this stage, we simulated games between RL agents and LLM agents to compare learning-based and reasoning-based action selection. The following tables present a comparative analysis of the performance metrics between the LLM-based agent and two selected DQN agents. The overall game (episode) number and games that reached turn cap are given in Table 5.9. Also, key indicators such as total turns, bluff success rates, and challenge effectiveness are used to evaluate agent behaviors in the given table 5.10.

Table 5.9: LLM agent evaluation - RL and LLM agents matchup: Total number of games simulated and number of games that reached the turn limit.

Metric	Value
Games Played	250
Games (100 Turns)	0

Table 5.10: Performance metrics of LLM and RL agents.

Metric	LLM_Agent	RL_Agent_3	RL_Agent_0
Total Turns	1695	1744	1756
Move Turns	1468	1588	1600
Total Bluffs	1188	1035	1035
Forced Bluffs	148	35	27
Successful Bluffs	601	418	435
Challenge Turns	1384	1491	1531
Total Challenges	852	772	770
Successful Challenges	620	602	582
Avg Cards / Move	1.221	2.482	2.466

The LLM_Agent played an average of 1.221 cards per move, significantly lower than RL_Agent_0 which uses 2.466 cards per move and RL_Agent_3 with 2.482 cards per move. This lower efficiency may be contributing to the overall lower win rate, as it suggests more conservative or cautious play while playing aggressive bluff actions, but it is possibly a side effect of reasoning-driven decisions prioritizing risk minimization.

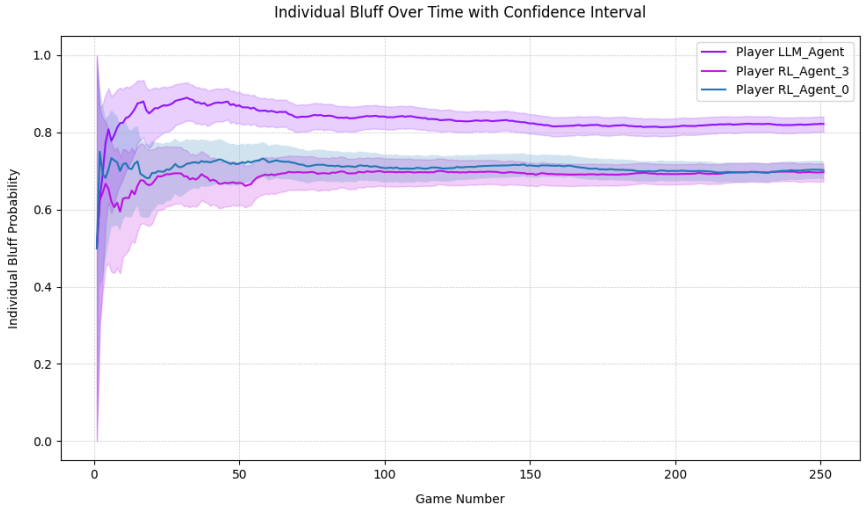

Figure 5.13: Bluff execution rates across bluff attempts for all agents in 2 RL- LLM agents mixed match-ups

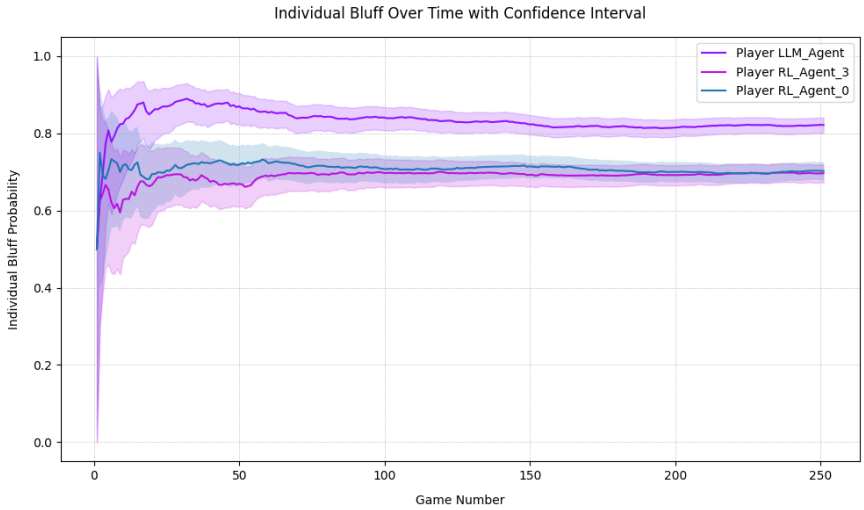

Figure 5.14: Successful bluff rates across bluff attempts for all agents in 2 RL- LLM agents mixed match-ups

For the action selection across games, we noticed that the LLM_Agent adopted the most aggressive action style among all agents, bluffing and challenging more often than

its RL counterparts. The LLM_Agent bluffed more often than the RL agents, with 1188 attempts, but it was not as successful in terms of deceptiveness as seen in Table 5.13 and 5.14 . It also had 166 forced bluffs, meaning it still chose to bluff even when it had matching cards, which suggests a riskier style of play.

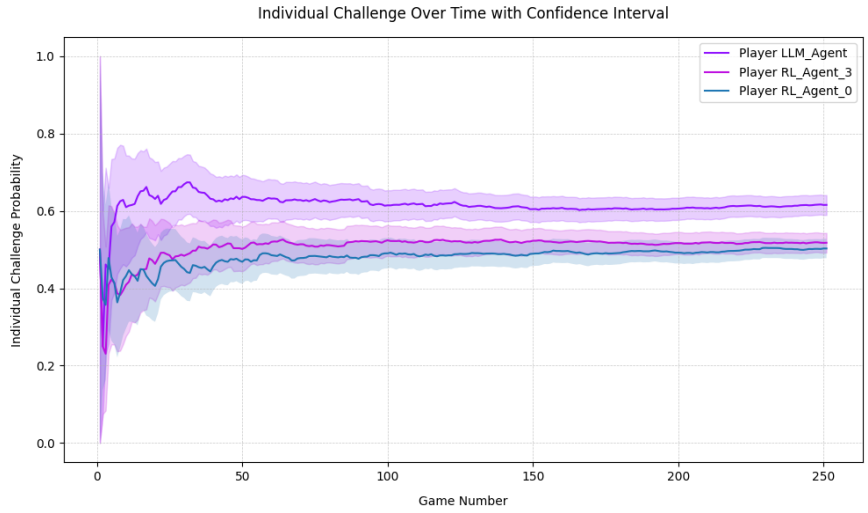

Figure 5.15: Challenge execution rates across bluff attempts for all agents in 2 RL-LLM agents mixed match-ups

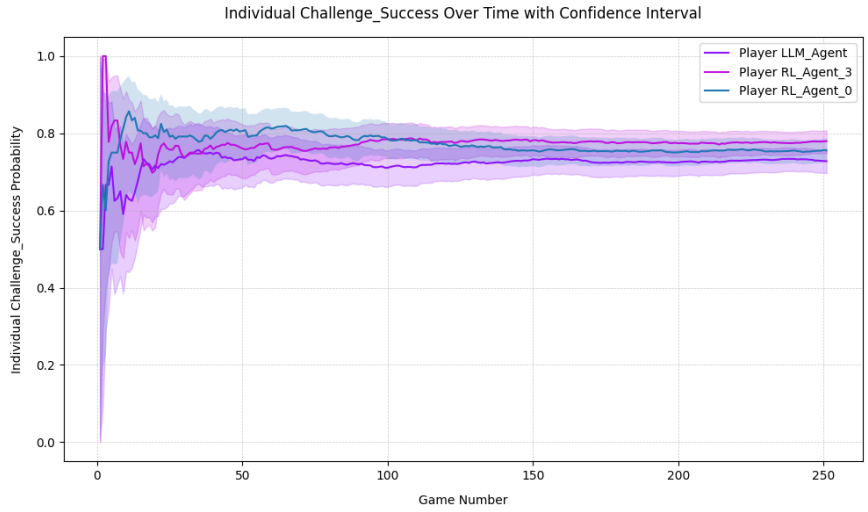

Figure 5.16: Successful challenge rates across bluff attempts for all agents in 2 RL-LLM agents mixed match-ups

Moreover, in terms of executing challenges, the LLM agent made 828 and got 593 right. But still, both RL agents managed similar or better results with fewer challenges. The graphs in figure 5.15 and 5.16 represent the success rates of challenge issues and

challenge execution rates, showing that the RL agents, especially RL_Agent_0, made more careful and consistent decisions when choosing to challenge.

While this shows high engagement and a willingness to take risks for LLM-Agent, its moderate but highest bluff success as 50.6% and relatively low challenge accuracy as 72.8% when we compare its opponents indicate a lack of precision in decision-making. However, when we examine its action individually, we can say that the LLM agent can suggest optimal actions but should make inferences in a more detailed way to analyze their opponent's behavior. Despite showing potential for flexible and dynamic gameplay, the LLM agent struggled to translate its frequent actions into consistently successful outcomes. In contrast, RL agents, especially RL_Agent_3, demonstrated more optimal and effective strategies, and outperformed reasoning-based models in terms of strategic efficiency and reliability in high-risk scenarios.

The win graphs show in Figures 5.17,5.18 and 5.19 that the LLM_Agent maintained competitiveness in early phases, but was eventually outperformed by both RL agents, especially RL_Agent_3. This is also evident from the average win rate curves: while RL_Agent_3 started with a strong advantage which is above 50%, its performance gradually declined, stabilizing around 37%. RL_Agent_0, on the other hand, remained more stable throughout, with a win rate fluctuating around 36%. In contrast, the LLM_Agent's performance in terms of being the winner showed slow improvements over time and reached a peak win rate of around 27%. However, it stayed at that level in the later stages and does not show further improvement in win rates. This shows it had some consistent decision-making, but not enough to outperform the RL agents.

While the LLM_Agent demonstrates the capacity to form structured play patterns and exhibits gradual improvement over time, its overall performance remains below that of learning-based agents. The agent's frequent bluff attempts, paired with only moderate success, indicate a tendency toward a more aggressive bluffing strategy, but it lacks consistent outcome prediction. These findings suggest that, although LLMs are capable of generalizing game behavior and performing human-like strategic reasoning, they currently do not match the consistent optimization and strategic exploitation achieved by reinforcement learning agents over turn-based, bluff environments.

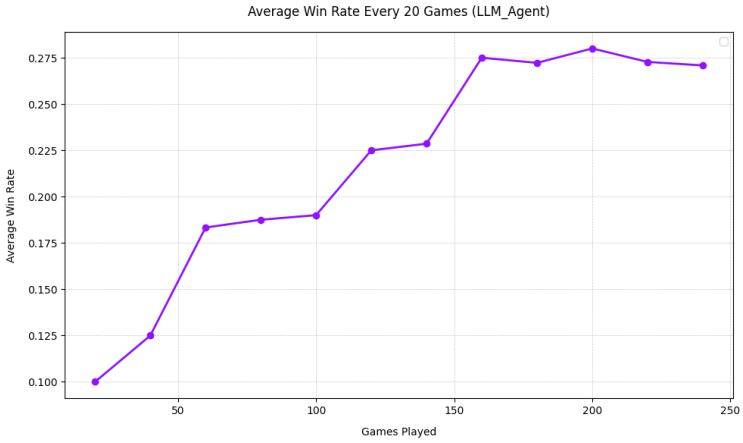

Figure 5.17: Average win per 20 levels for LLM_Agent

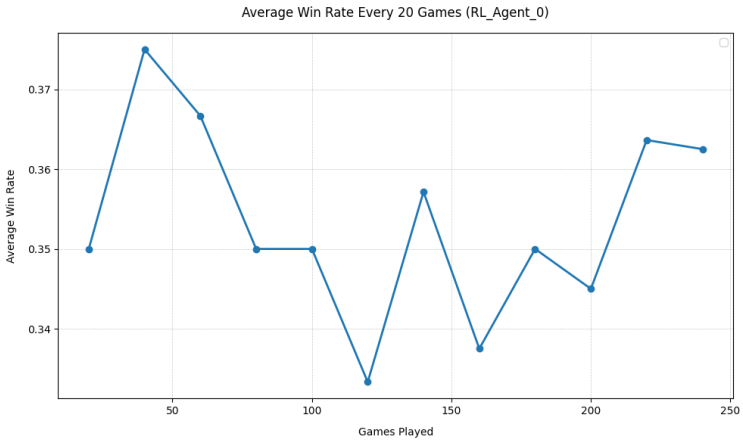

Figure 5.18: Average win per 20 levels for RL_Agent_0

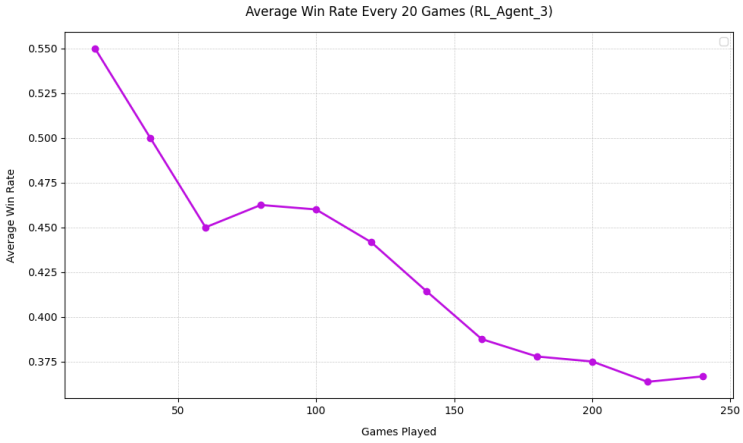

Figure 5.19: Average win per 20 levels for RL_Agent_3

5.1.5.3 Baseline strategy match-up for LLM agent's performance evaluation

In this step, we examine the comparison between baseline agents and LLM-agent. Overall game information about sample size and number of games with max turn in Table 5.11.

Table 5.11: LLM agent evaluation - Baseline and LLM agents matchup: Total number of games simulated and number of games that reached the turn limit.

Metric	Value
Games Played	250
Games (100 Turns)	0

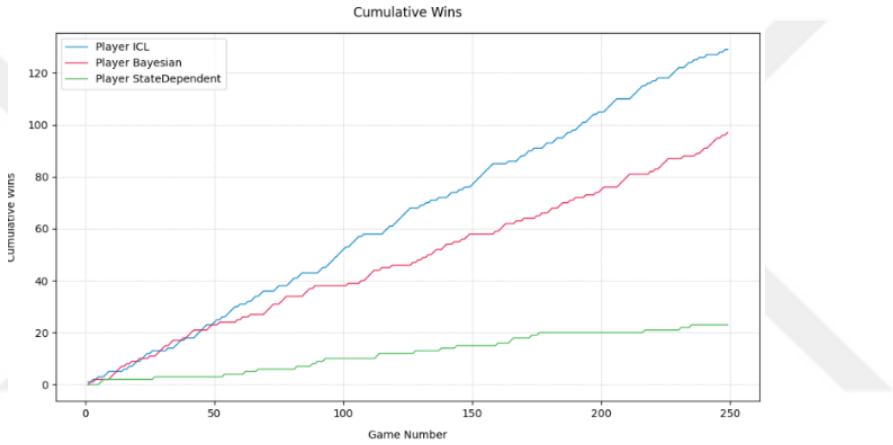

Figure 5.20: Win rates of LLM, Bayes and State-dependent agents match-up

LLM agent exhibited the highest win rate as 51.6% among all agents (Table 5.20), which suggests that its aggressive and proactive play style paid off in multi-agent settings. The Table 5.21 represents actions selection rates and their success rates for each agent.

	Baseline State Dependent Agent	Baseline Bayes Agent	LLM Agent
Win Rate	9,20%	38,80%	51,60%
Challenge Rate	70,20%	68,40%	77,70%
Challence Sucess	34,80%	72,00%	97,80%
Bluff Rate	71,80%	76,00%	66,20%

Figure 5.21: Action selection and success frequencies of LLM, Bayes and State-dependent agents match-up

It also had the highest challenge rate with 77.7% and the highest challenge success as 97.8%, showing that its decisions were not only frequent but also highly accurate. While its bluff rate as 66.2% was slightly lower compared to its opponents since the Baseline Bayes Agent has 76.0% and the State Dependent Agent as 71.8% bluff rate, the LLM's overall strategy appears more balanced and opportunistic, choosing to bluff selectively while maintaining strong challenge pressure. In contrast, the Baseline Bayes Agent showed relatively high performance with a 38.8% win rate and solid challenge success whereas the State Dependent Agent struggled with both win rate and challenge accuracy, indicating vulnerability against more adaptive strategies with prior knowledge. Overall, the LLM's ability to combine calculated bluffing with highly successful challenges seems to be the key factor behind its superior performance.

LLM's high challenge rate can be understood from the structure of its strategy: Because the LLM agent evaluates game history and prior knowledge, it was able to challenge opponents with greater confidence. Compared to other agents, it adopted a more cautious yet aggressive play style, applying constant pressure by challenging its opponents; it gained an advantage by more frequently catching opponents' mistakes. This demonstrates that a high challenge rate is not a random or unnecessary risk, but rather a strategic and data-driven choice. This might also relate to the play style of its opponents, since they use the same number of cards for both bluff and truthful actions, and since truthful actions have to be less than bluff actions, LLM suggests an action that gives a better chance to win by catching their bluff as the best action.

In short, while LLM agent's decision to challenge almost every turn may seem excessive at first glance, this choice stems from a high level of accuracy and strong decision-making. This allowed LLM agent to become the game's leader and win the game despite its lower bluffing success, as we can see in the win rate graph in Figure 5.20.

5.1.5.4 Mixed strategy match-up for LLM agent's performance evaluation

To further evaluate the reasoning-based performance of the LLM_Agent, we conducted an additional 250-game simulation where it was matched against RL_Agent_3 and a rule-based Baseline_Bayes agent. Since these agents are the most successful ones

among their own category, this simulation will evaluate LLM agent's performance against the most successful opponents. This setting enables a direct comparison between reasoning-driven behavior, learning-based decision-making, and static probability-guided bluffing. The overall game metrics are shown in Table 5.12, and detailed behavioral outcomes are provided in Table 5.13.

Table 5.12: LLM agent evaluation - mixed matchup: Total number of games simulated and number of games that reached the turn limit.

Metric	Value
Games Played	250
Games (100 Turns)	0

Table 5.13: Performance metrics of agents in mixed-strategy evaluation setup.

Metric	LLM	RL-3	Bayes
Total Turns	2210	2265	2217
Move Turns	1980	1953	1994
Total Bluffs	1531	1263	1089
Forced Bluffs	166	15	31
Successful Bluffs	551	362	641
Challenge Turns	1782	2011	1884
Total Challenges	828	999	1321
Successful Challenges	593	655	1081
Avg Cards / Move	1.186	2.576	1.202

The LLM_Agent played an average of 1.186 cards per move, which is quite close to Baseline_Bayes at 1.202, and both were noticeably lower than RL_Agent_3 with 2.576. Despite playing fewer cards, the LLM_Agent bluffed far more aggressively than the other two, with 1531 bluff attempts which is significantly more than RL_Agent_3 (1263) and Baseline_Bayes (1089). However, this aggressive approach didn't lead to high bluff accuracy.

Meanwhile, based on the figures 5.22 - 5.23, LLM_Agent had the lowest bluff success rate at 35.9%, while Bayes reached 58.8%. Still, it outperformed RL_Agent_3, whose bluff success was even lower at 28.6%.

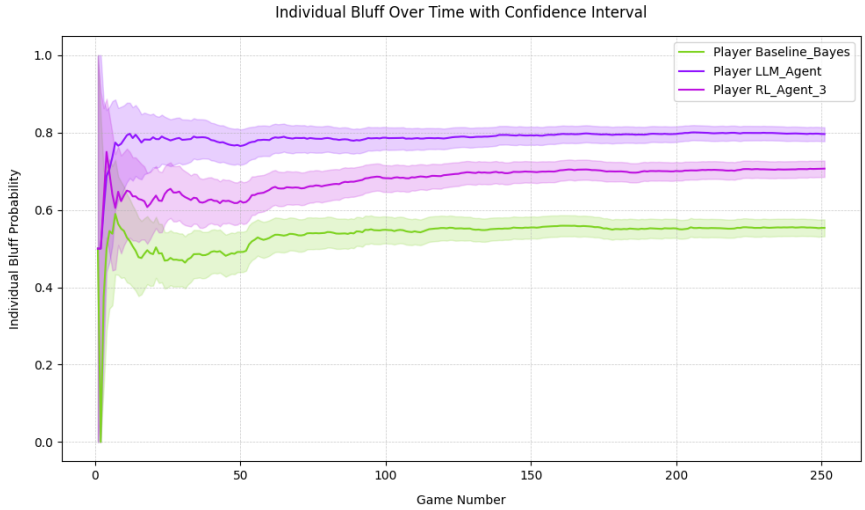

Figure 5.22: Bluff execution rates across bluff attempts for all agents in RL-LLM-Baseline mixed match-up

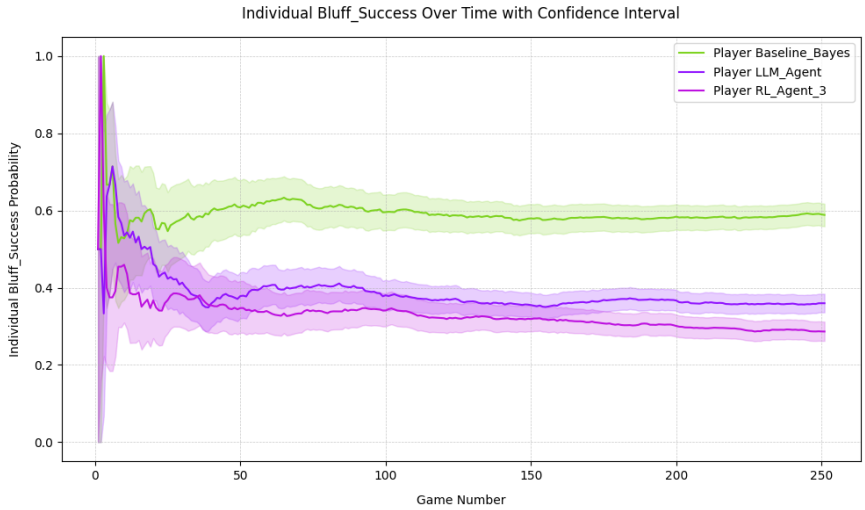

Figure 5.23: Successful bluff rates across bluff attempts for all agents in RL-LLM-Baseline mixed match-up

This result might suggest that even though RL_Agent_3 shows stronger performance in its matchup with RL and LLM agents, its high-risk play style, which is reflected in the higher average number of cards per move, created an environment where more cautious agents like the LLM agent were able to improve their action selection by relying on both prior knowledge and current game memory.

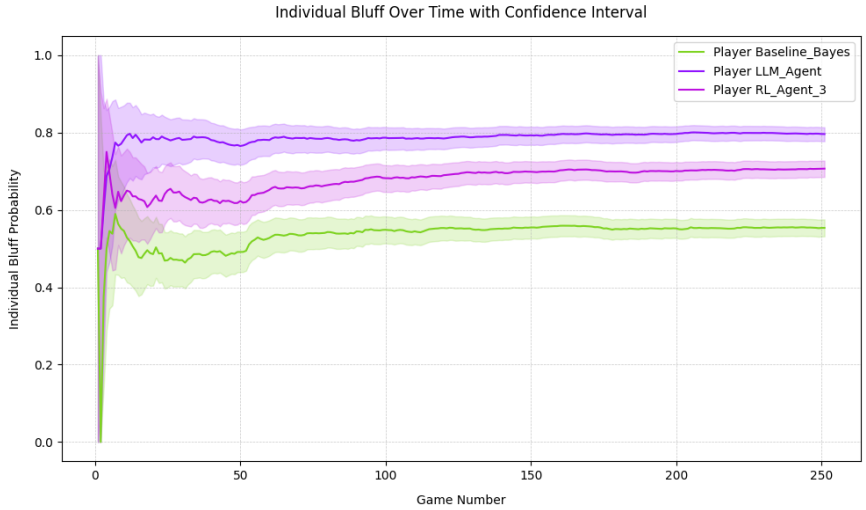

Figure 5.24: Challenge execution rates across bluff attempts for all agents in RL-LLM-Baseline mixed match-up

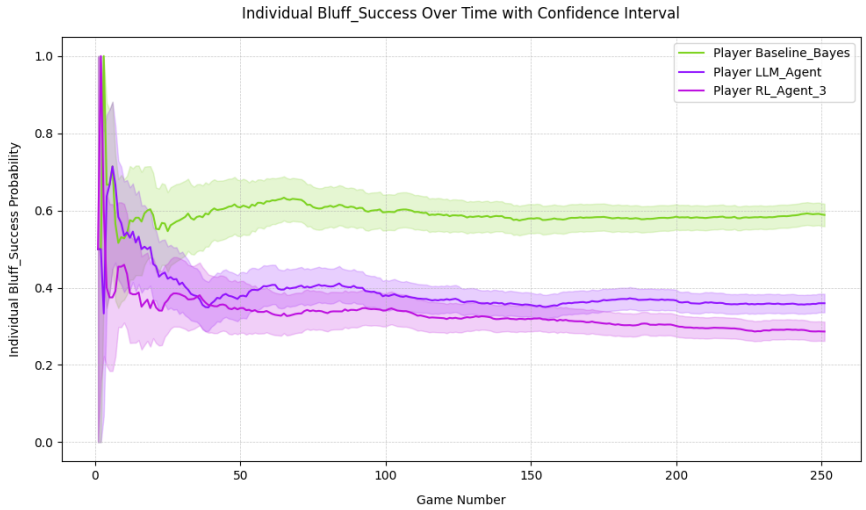

Figure 5.25: Successful challenge rates across bluff attempts for all agents in RL-LLM-Baseline mixed match-up

The LLM agent also made fewer challenges as 828 than Bayes as 1321 and RL_Agent_3 as 999, but again showed better precision than RL_Agent_3, with a challenge success rate of 71.6% compared to 65.5%. These metrics are represented in figures 5.24 and 5.25. So, even though the LLM agent's actions weren't always optimal, it showed better effectiveness than RL_Agent_3 in this setting.

Figures 5.26, 5.27, and 5.28 illustrate the win rate of each agent as the average win rate every 20 games across the match-up. According to the result, Baseline_Bayes

clearly performed the best, steadily improving and finishing with a win rate over 60%. The LLM_Agent started with a win rate close to 40%, then dropped and stabilized around 30%. Even though this is far from Bayes' level, it is still noticeably better than RL_Agent_3, which struggled throughout the simulation and stayed around 10%. Compared to the earlier 2 RL–1 LLM agents setup, where the LLM agent lagged behind both learning-based opponents, here it consistently outperformed RL_Agent_3 in almost all metrics—win rate, bluff success, and challenge accuracy.

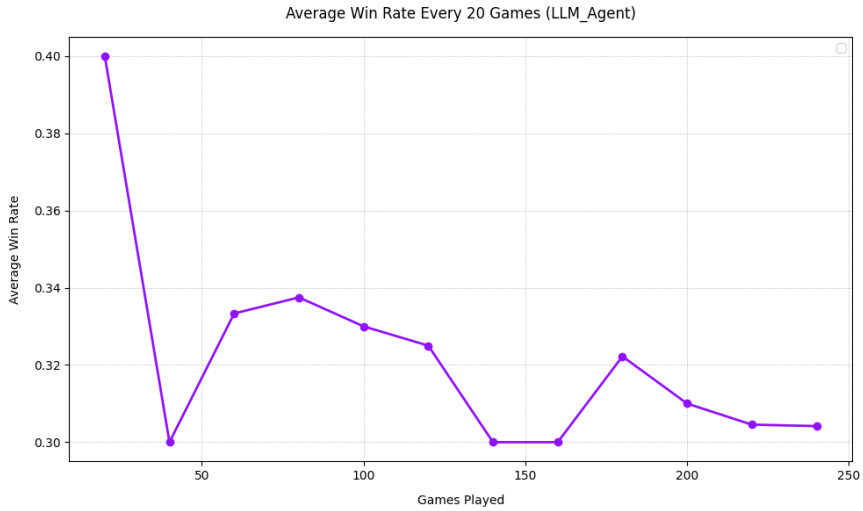

Figure 5.26: Average win per 20 levels for LLM_Agent

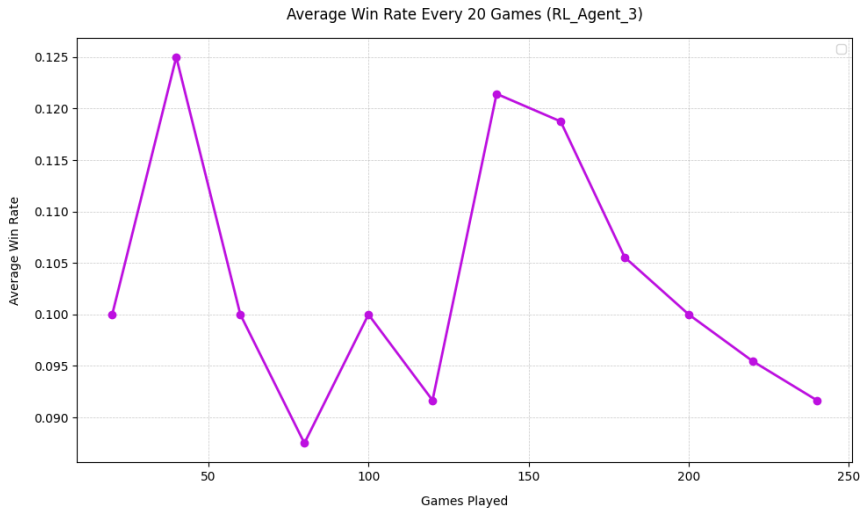

Figure 5.27: Average win per 20 levels for RL_Agent_0

These results suggest that while the LLM_Agent still falls short of outperforming a well-structured rule-based model like Bayes, it is capable of maintaining competitive

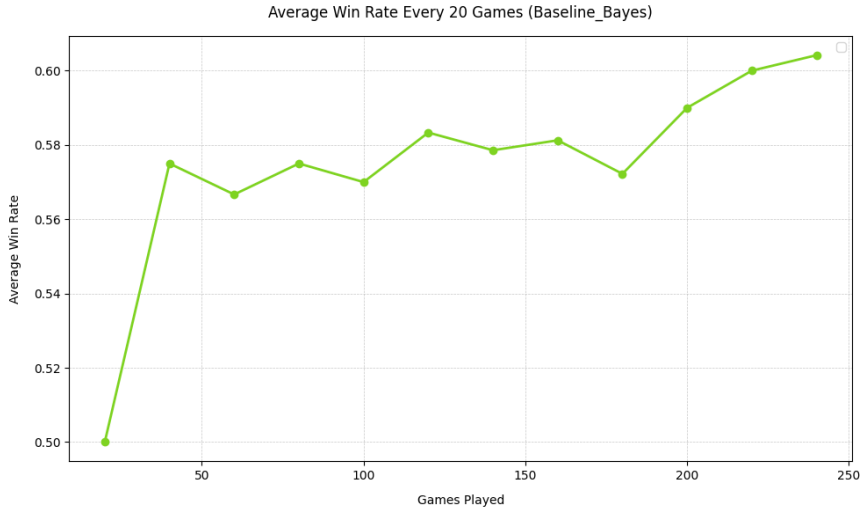

Figure 5.28: Average win per 20 levels for Baseline_Bayes

performance and even outperforming weaker learning agents like RL_Agent_3. This may be because the Bayes agent uses prior history and always plays more conservatively than others when uncertainty is high. So even though the LLM agent's actions weren't always optimal, it showed better control and effectiveness than RL_Agent_3 in this setting. Also, this result demonstrates that reasoning-based agents with in-context learning can hold their ground in mixed agent environments.

6. CONCLUSION

In this thesis, a simulation environment for a bluff-based card game is designed and implemented to evaluate different agent decision-making strategies. The goal is defined as comparing four distinct agent types which are Random, Rule-Based, Reinforcement Learning (RL), and Large Language Model (LLM) agents, under the same game environment with stochastic conditions using clearly defined metrics. These simulations are conducted to capture both performance and behavioral tendencies in a deceptive, partially observable setting.

The baseline match-up results are shown, where the Bayesian agent is observed to perform best among the baseline agents since the use of prior game statistics is allowed for more informed bluff detection and execution, and both the Random and State-Dependent agents are outperformed in terms of challenge accuracy, bluff success, and win frequency. These comparisons are confirmed to demonstrate the effectiveness of statistical inference in limited-information environments, and a reference point for evaluating learning-based and reasoning-based agents is established.

Next, RL agents are trained through both baseline-oriented and self-play settings. In particular, RL_Agent_3 and RL_Agent_0, which are trained via self-play and baseline-oriented methods respectively, are described as the most consistent performers across various metrics, as high bluff and challenge success rates are maintained and stable win rates in test-time evaluations are observed. The value of self-play for developing generalized strategies in uncertain and dynamic game conditions is reinforced.

For the LLM agent, while high-risk actions are frequently selected and detailed justifications are produced through in-context reasoning, the overall performance is observed to be less stable than that of the RL agents when the win rates are considered. The LLM agent's behavior is reflected as a blend of cautious play in card usage and aggressive play in bluff frequency. It is found to fall behind both RL agents in the 2

RL–LLM setting but is shown to perform more competitively in the mixed match-up with RL and Bayesian opponents. Notably, RL_Agent_3 is outperformed in challenge and bluff success rates and a consistent win rate around 30% is maintained.

These findings are suggested to show that LLM-based agents, which reason via prompts defined in a cost-efficient but detailed way, are able to be adapted to deceptive game contexts and strategic awareness through structured prompting is demonstrated, even without training or fine-tuning. However, their performance is observed to still lack the precision and optimization achieved by agents trained with explicit reward signals, so even if turn-based action results are succeeded, it is not found to be effective enough to make it an overall winner. To increase its reasoning method, the initial condition of the game and memory of the game are suggested to be described in a more detailed way. That said, the LLM's ability to justify actions in natural language is offered as a major advantage in terms of transparency and interpretability, which is considered especially valuable in human-agent interaction scenarios or evaluation settings.

In conclusion, a comprehensive comparison between random, probabilistic, learning-based, and reasoning-based agent types within a deceptive card game framework is presented in this study. While reinforcement learning and Bayesian inference for decision making are identified as the most effective approaches in terms of strategic success and win rate, the LLM agent is offered as a promising direction for reasoning-driven decision making and inference based on prompt structure.

6.1 Future Work

As future work, combining reinforcement learning and in-context reasoning capabilities of large language models (LLMs) may lead to the development of agents that are both effective for multi-agent game environments. This integration can be explored through hybrid agent architectures or collaborative prompting schemes, where LLMs assist or guide RL agents in real-time decision making. In addition to that, we can implement reward shaping methods and more sophisticated prompt structures by incorporating natural behavior patterns to generate more optimal strategies that lead LLM to reason about not only turn-specific actions but also win conditions by analyzing the opponents' behavior in a more comprehensive way.

Additionally, expanding the sample size and creating more detailed state representation would contribute to more reliable evaluation results and also, it can help us create a generalized algorithm for different numbers of match-ups and generalize the game environment.

Combining the structured learning capabilities of RL with the flexible reasoning capabilities of LLMs can provide high-performance training that can make better inferences and better interpret action and game flow in different game scenarios.

REFERENCES

[1] **Cowling, P.I.**, **Whitehouse, D. and Powley, E.J.** (2015). Emergent bluffing and inference with Monte Carlo tree search, *2015 IEEE conference on computational intelligence and games (CIG)*, IEEE, pp.114–121.

[2] **Luo, Q. and Tan, T.P.** (2023). RARSMSDou: Master the game of DouDiZhu with deep reinforcement learning algorithms, *IEEE Transactions on Emerging Topics in Computational Intelligence*, *8*(1), 427–439.

[3] **Huang, C.**, **Cao, Y.**, **Wen, Y.**, **Zhou, T. and Zhang, Y.** (2024). PokerGPT: An End-to-End Lightweight Solver for Multi-Player Texas Hold'em via Large Language Model, *arXiv preprint arXiv:2401.06781*.

[4] **Guo, J.**, **Yang, B.**, **Yoo, P.**, **Lin, B.Y.**, **Iwasawa, Y. and Matsuo, Y.** (2023). Suspicion-agent: Playing imperfect information games with theory of mind aware gpt-4, *arXiv preprint arXiv:2309.17277*.

[5] **Duan, J.**, **Zhang, R.**, **Diffenderfer, J.**, **Kailkhura, B.**, **Sun, L.**, **Stengel-Eskin, E.**, **Bansal, M.**, **Chen, T. and Xu, K.** (2025). GTBench: Uncovering the Strategic Reasoning Capabilities of LLMs via Game-Theoretic Evaluations, *Advances in Neural Information Processing Systems*, *37*, 28219–28253.

[6] **Wu, H.**, **Liu, X.**, **Wang, Y. and Zhao, H.** (2024). Instruction-Driven Game Engine: A Poker Case Study, *arXiv preprint arXiv:2410.13441*.

[7] **Hu, S.**, **Huang, T.**, **Ilhan, F.**, **Tekin, S.**, **Liu, G.**, **Kompella, R. and Liu, L.** (2024). A survey on large language model-based game agents, *arXiv preprint arXiv:2404.02039*.

[8] **Zeek and Ben**. *Building an LLM-based assistant for the game of Diplomacy*, `https://jataware.com/projects/react-diplomacy/`, accessed: 2025-02-27.

[9] **Brown, T.B.**, **Mann, B.**, **Ryder, N.**, **Subbiah, M.**, **Kaplan, J.**, **Dhariwal, P.**, **Neelakantan, A.**, **Shyam, P.**, **Sastry, G.**, **Askell, A.** *et al.* (2020). Language models are few-shot learners, *Advances in Neural Information Processing Systems*, *33*, 1877–1901.

[10] **Vaswani, A.**, **Shazeer, N.**, **Parmar, N.**, **Uszkoreit, J.**, **Jones, L.**, **Gomez, A.N.**, **Kaiser, and Polosukhin, I.** (2017). Attention is all you need, *Advances in Neural Information Processing Systems*, volume 30.

[11] **Han, W.**, **Zhang, S.S.**, **Wang, X.**, **Chi, E.H.**, **Le, Q.V.**, **Zhou, D. and Wei, J.** (2023). Explaining in-context learning as implicit Bayesian inference, *arXiv preprint arXiv:2309.00661*.

[12] **OpenAI** (2024). *GPT-4.1 Prompting Guide*, `https://cookbook.openai.com/examples/gpt4-1_prompting_guide`, accessed: 2025-05-25.

[13] **Wei, J.**, **Wang, X.**, **Schuurmans, D.**, **Bosma, M.**, **Ichter, B.**, **Xia, F.**, **Chi, E.H.**, **Le, Q.V. and Zhou, D.** (2022). Chain of thought prompting elicits reasoning in large language models, *arXiv preprint arXiv:2201.11903*.

[14] **Kojima, T.**, **Gu, S.S.**, **Reid, M.**, **Matsuo, Y. and Tanaka, Y.** (2022). Large language models are zero-shot reasoners, *arXiv preprint arXiv:2205.11916*.

[15] **Zhao, W.X.**, **Shao, Y.**, **Yan, R.**, **Yatskar, M.**, **Ma, J.R.** *et al.* (2023). A survey of hallucination in natural language generation, *arXiv preprint arXiv:2302.06476*.

[16] **Mnih, V.**, **Kavukcuoglu, K.**, **Silver, D.**, **Rusu, A.A.**, **Veness, J.**, **Bellemare, M.G.**, **Graves, A.**, **Riedmiller, M.**, **Fidjeland, A.K.**, **Ostrovski, G.** *et al.* (2015). Human-level control through deep reinforcement learning, *nature*, *518*(7540), 529–533.

[17] **van Hasselt, H.**, **Guez, A. and Silver, D.** (2016). Deep reinforcement learning with double Q-learning, *Proceedings of the AAAI Conference on Artificial Intelligence*, *30*(1).

[18] **Wang, Z.**, **Schaul, T.**, **Hessel, M.**, **Van Hasselt, H.**, **Lanctot, M. and De Freitas, N.** (2016). Dueling network architectures for deep reinforcement learning, *International Conference on Machine Learning*, PMLR, pp.1995–2003.

[19] **Schaul, T.**, **Quan, J.**, **Antonoglou, I. and Silver, D.** (2016). Prioritized experience replay, *International Conference on Learning Representations (ICLR)*.

[20] **Henderson, P.**, **Islam, R.**, **Bachman, P.**, **Pineau, J.**, **Precup, D. and Meger, D.** (2018). Deep reinforcement learning that matters, *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1).

[21] **Gamezy** (n.d.). *Bluff Card Game: Know How to Play Bluff Games & Its Rules*, `https://www.gamezy.com/card-games/bluff-card-game/#some-modified-bluff-card-games`, `https://www.gamezy.com/card-games/bluff-card-game/#some-modified-bluff-card-games`, accessed: 2025-05-20.

[22] **wikiHow contributors** (n.d.). *How to Play Bluff*, `https://www.wikihow.com/Play-Bluff`, `https://www.wikihow.com/Play-Bluff`, accessed: 2025-05-20.

[23] **Paszke, A.**, **Gross, S.**, **Massa, F.**, **Lerer, A.**, **Bradbury, J.**, **Chanan, G.**, **Killeen, T.**, **Lin, Z.**, **Gimelshein, N.**, **Antiga, L.**, **Desmaison, A.**, **Kopf, A.**, **Yang, E.**, **DeVito, Z.**, **Raison, M.**, **Tejani, A.**, **Chilamkurthy, S.**, **Steiner, B.**, **Fang, L.**, **Bai, J. and Chintala, S.** (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library, *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, pp.8024–8035, `https://pytorch.org`.

[24] **Li, Y.** (2017). Deep reinforcement learning: An overview, *arXiv preprint arXiv:1701.07274*.

[25] **Sutton, R.S.**, **Barto, A.G.** *et al.* (1998). *Reinforcement learning: An introduction*, volume 1, MIT press Cambridge.

[26] **Sokolowski, J.A.** (2010). Monte Carlo Simulation, *Modelling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains, Wiley & Sons Inc., New Jersey*, 131–145.

[27] **Çakır, O.** (2024). *İstatistik Fizik ve Termodinamik Ders Notları (FİZ304)*, ankara Üniversitesi Kütüphanesi Açık Ders Malzemeleri.

[28] **Gelman, A.**, **Carlin, J.B.**, **Stern, H.S.**, **Dunson, D.B.**, **Vehtari, A. and Rubin, D.B.** (2013). *Bayesian Data Analysis (3rd Edition)*, Chapman & Hall/CRC.

[29] **Murphy, K.P.** (2012). *Machine Learning: A Probabilistic Perspective*, MIT Press.

[30] **Albert, J. and Hu, J.** (2020). *Probability and Bayesian Modeling*, Texts in Statistical Science, Chapman & Hall/CRC, Boca Raton, FL, `https://bayesball.github.io/BOOK/`, accessed: 14 March 2025.

[31] **Korb, K.B.**, **Nicholson, A.E. and Jitnah, N.** (1999). Bayesian Poker, *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI)*, pp.343–350.

[32] **Chen, B. and Ankenman, J.** (2006). *The Mathematics of Poker*, ConJelCo Press.

APPENDICES

APPENDIX A : Mathematical Preliminaries

Appendix A: Mathematical Preliminaries

This chapter presents the foundational probability and statistical concepts needed to model uncertainty and decision-making in games.

1.1 Concept of Probability

The theory of probability offers a mathematical method to establish the rules for each of its games to ensure that the odds are in their favor, which is where probability comes into play. [26]

Basically, probability is a measure that quantifies the likelihood of an event occurring. It is expressed as a number between 0 and 1, where 0 indicates impossibility and 1 indicates certainty. A simple space lists all possible outcomes of a random experiment, denoted by X. A random variable is a function that assigns a real number to each outcome in the sample space. Since a trial results in an elementary event x_i, from a set of events $X = \{x_i | i = 1, ..., N\}$, the probability of the occurrence of each event can be shown as

$$P(x_i) \tag{A.1}$$

1.1.1 Probability axioms and properties

The axioms of probability, as formulated by Kolmogorov, define the fundamental properties of probability measures.

1. **Non-negativity Axiom**:

   This axiom states that the probability of any event is never negative. For any event x_i, we have:

   $$P(x_i) \geq 0 \tag{A.2}$$

2. **Normalization Axiom**:

   If the set X is countable, then P(x) is called discrete random variable and it formalized as

   $$\sum_{i=1}^{N} P(x_i) = 1, \quad P(x_i)\, \varepsilon\, (0.1) \tag{A.3}$$

3. **Countable Additivity Axiom**:

   If you have a sequence of mutually exclusive (disjoint) events $x_1, x_2, x_3, \ldots$ (i.e.,

events that cannot occur simultaneously), then the probability of their union is the sum of their individual probabilities:

$$P\left(\bigcup_{i=1}^{\infty} x_i\right) = \sum_{i=1}^{\infty} P(x_i) \tag{A.4}$$

1. **Property 1: Monotonicity**

   If $x_i \subset y_j$, then

   $$P(x_i) \leq P(y_j). \tag{A.5}$$

   Since $x_i \subset y_j$, the set y_j can be written as the union of x_i and the set difference $y_j \setminus x_i$:

   $$y_j = x_i \cup (y_j \setminus x_i). \tag{A.6}$$

   Because x_i and $y_j \setminus x_i$ are mutually exclusive,

   $$P(y_j) = P(x_i) + P(y_j \setminus x_i). \tag{A.7}$$

   The non-negativity axiom tells us that $P(y_j \setminus x_i) \geq 0$. Thus,

   $$P(y_j) \geq P(x_i), \tag{A.8}$$

   which completes the proof.

2. **Property 2: Boundedness**

   For any event x_i,

   $$P(x_i) \leq 1. \tag{A.9}$$

   Recognize that any event x_i is a subset of the sample space S (i.e., $A \subset S$). Thus,

   $$P(x_i) \leq P(X). \tag{A.10}$$

   By the normalization axiom,

   $$P(X) = 1. \tag{A.11}$$

   Hence,

   $$P(x_i) \leq 1. \tag{A.12}$$

   In addition to these properties we can also write probability of the empty. Since the empty set $\emptyset$ contains no outcomes, by considering that

   $$P(\emptyset) + P(X) = P(X), \tag{A.13}$$

   therefore,

   $$P(\emptyset) = 0. \tag{A.14}$$

1.1.2 Counting methods

1. Factorial: Suppose we have n number of element in a set. We can find how many ways to arrange these elements in the set by using factorial:

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 1 \quad \text{(A.15)}$$

2. Permutation: Suppose that we have n number of elements and we want to choose r number of elements from it where $r = [0,n]$. How many ways to choose r elements from a set with size n is evaluated as follow and called permutation:

$$P(n,r) = \frac{n!}{(n-r)!} \quad \text{(A.16)}$$

3. Combination: Suppose that we have n number of elements and we want to choose k number of elements from it where $k = [0,n]$. How many ways to choose k elements from this set without considering the order of selection is evaluated as follow and called combination:

$$C(n,k) = \frac{n!}{k!(n-k)!} \quad \text{(A.17)}$$

4. Number of Subset: The number of different subsets (including the empty set and set itself) is determined by the formula below:

$$2^n \quad \text{(A.18)}$$

Let us illustrate these concepts by giving an example with different aspects of the game of poker. A standard deck has 52 cards. The number of ways to arrange (or shuffle) the entire deck is given by the factorial of 52:

$$52! = 52 \times 51 \times 50 \times \cdots \times 1$$

Suppose that we consider dealing five cards from a deck.

$$\binom{52}{5}$$

For a standard poker game, the order in which the cards are held does not matter. To count the number of unique hands of a three-of-a-kind, we use combinations. For a three-of-a-kind, there must be 3 cards with same rank value and 2 cards with different rank value. Since there consists of 4 suits and 13 ranks in a deck, we represent how many ways to occur three of a kind in a hand as below:

$$\binom{13}{1} \times \binom{4}{3} \times \binom{12}{2} \times \binom{4}{1}^2$$

For any given hand, the number of different subsets (selections of cards, including the empty set and the full hand) is determined by the formula 2^n, where n is the number of cards in the hand. For a 5-card hand:

$$2^5 = 32$$

This number represents every possible combination of cards that that can be chosen from the hand, regardless of order.

1.1.3 Classical view of probability

The classical view of probability explains all possible outcomes for an event have the same probability which defined as the ratio of the number of favorable outcomes to total number of outcomes. Thus, we can say that all outcomes in a given experiment are equally likely. To illustrate this idea, consider an experiment like coin tossing or rolling dice. All outcomes can be divided into subgroups according to what they are, for example while there are 2 possible outcomes of tossing a coin, there are 6 possible outcomes of rolling a dice. Suppose we express the event with A, all possible outcomes as a set named $X = \{x_1, x_2, \cdot, x_n\}$ where $n = |X|$, and ways to occur of an outcome donated by m, (for most case (random events), each outcome is unique and equally likely, therefore m = 1.). Then, we can write the probability of occurring of an outcome x_i as given below:

$$P(x_i) = \frac{m}{n} \tag{A.19}$$

According to the equation above, then the probability of occurring heads in a coin tossing is:

$$P(heads) = 1/2 \tag{A.20}$$

since there are only 2 possible outcomes and they are equally likely. However, some cases, the probabilities of occurring events in the sample space can differ. For example, every poker hand has a different probability of occurring, and this determines the value of the hand. We showed ways to choose 5 cards out of 52 cards and getting three-of-a-kind (AAABC) combinations from 52 cards in the previous section. So, by using them, we can write the probability of getting three of a kind as below:

$$P(x_i = AAABC) = \frac{\binom{13}{1} \times \binom{4}{3} \times \binom{12}{2} \times \binom{4}{1}^2}{\binom{52}{5}}$$

1.1.4 Frequency view of probability

The frequentist interpretation of probability applies whether outcomes are equally likely or not. This approach is suitable when a random experiment can be repeated multiple times under consistent conditions. To illustrate this concept, consider an experiment like tossing coins or rolling dice and define N as all possible outcomes of the experiment.

As we mentioned, there are two and six possible outcomes, respectively. Outcomes of each trial can be described as an event, and if a previous event does not affect the current one, then they are said to be independent. The probability of occurrence of an independent event is written as follows

$$P_s = \lim_{N\to\infty} \frac{N_s}{N}, \quad P_s \varepsilon (0.1) \tag{A.21}$$

Based on the frequentist approach, we can demonstrate the probability of getting heads by tossing a coin infinitely many times and the probability of getting three of a kinds by dealing the 52 cards infinitely many times.

1.1.5 Bayesian view of probability

Consider a coin tossing game; this time suppose the player is not sure about whether the coin is fair or unfair. Since the player cannot repeat the experiment infinitely to determine whether it's fair or fraudulent, the probability calculation is done with constraints as beliefs that the player obtains based on the characteristics of the coin, such as its appearance and movement, and also observed outcomes of performed experiments, giving a personal view of probability. If the results of the experiment are not equally probable or it is not feasible to replicate the experiment numerous times under similar conditions, the subjective perspective of probability becomes relevant for making estimations. This approach provides a flexible framework for understanding and quantifying uncertainty in a wide range of scenarios.

For example, a player is in a poker game and holds a pair of Queens. The opponent makes a large bet on the river. The player does not know whether the opponent has a strong hand or is bluffing, so they must estimate the probability subjectively by using observations such as the player thinks the opponent has bet aggressively and this opponent has bluffed in 70% of similar situations in past games. So, the player believes the opponent is more likely to bluff in this game too based on experience and observation.

1.2 Conditional Probability

In the case more than one outcome can occur at each trial, which is called multiple random variables, such as two sets of possible events at each trial $X = \{x_i | i = 1, ..., N_X\}$ and $Y = \{y_j | j = 1, ..., N_Y\}$, the corresponding probability distribution is joint probability, which is

$$\sum_{i=1}^{N_X} \sum_{j=1}^{N_Y} P(x_i, y_j) = 1 \tag{A.22}$$

The probability of having x and any value of y is defined as marginal probability and is basically defined as the sum over the events of the other set,

$$P_X(x) = \sum_y P(x, y) \tag{A.23}$$

and this can be normalized to one. In the case the events have already occurred, the probability of the other is called conditional probability

$$P(x|y) = \frac{P(x,y)}{P_Y(y)} \tag{A.24}$$

and if X and Y are independent random variables then $P(x,y)$ can be calculated separately as $P_X(x)P_Y(y)$ and the equation above satisfies that $P(x|y) = P_X(x)$.

To generalize the probability equation, we can write a multi-dimensional random variable and illustrate that with a rolling dice process. Let's consider 2 rolling dice, there are 6 outcomes for both dice and each of them is independent. In general, possible outcomes can be written as,

$$\begin{aligned} X &= \{x_i | i = 1, ..., N_X\} = \{1,2,3,4,5,6\} \\ Y &= \{y_j | j = 1, ..., N_Y\} = \{1,2,3,4,5,6\} \\ f &: (X,Y) \to Z : (x,y) \to z = f(x,y) \end{aligned} \tag{A.25}$$

f is the function defined on X,Y which determines the corresponding event.

For example, to find the probability of rolling dice results with a total greater than 2, we should find the sum of dice results for each rolling. So, we can define a new relation as

$$\begin{aligned} Z &= \{f(x_i, y_j) | i = 1, ..., N_X; j = 1, ..., N_Y\} \\ &= \{2,3,4,...,12\} \end{aligned} \tag{A.26}$$

Moreover, its probability distribution,

$$\begin{aligned} P_Z(z) &= \sum_x \sum_y P(x,y) \delta_{z,f(x,y)} \\ &= \sum_x \sum_y P(x)P(y) \delta_{z,f(x,y)} \\ P(2) &= \frac{1}{6}\frac{1}{6}1 = \frac{1}{36} \end{aligned} \tag{A.27}$$

where $\delta_{z,f(x,y)}$ is the Dirac-Delta function, which means the function yields 1 if the result equals 2, otherwise 0.

We can also represent this probability calculation in set-notation as follows:

$$P_Z(z) = P(x_i \cap y_j) = \begin{cases} P(y_j|x_i) \cdot P(x_i) & \text{if } x_i, y_j \text{ are dependent,} \\ P(x_i) \cdot P(y_j) & \text{if } x_i, y_j \text{ are independent} \end{cases} \tag{A.28}$$

where $P_Z(z) = P(x_i|y_j)$ and $P(x_i \cap y_j)$ is also called joint probability.

1.2.1 Bayes' rule

Bayes' theorem states that if x_i and y_j are dependent, the conditional probability can be rewritten as:

$$P(x_i|y_j) = \frac{P(y_j|x_i)P(x_i)}{P(y_j)}. \tag{A.29}$$

This formulation highlights how prior knowledge about x_i affects our belief about y_j, making Bayes' rule a fundamental concept in probability theory and inference.

Let make an example to explain Bayes' Rule as follows: Suppose a player is uncertain whether a six-sided die is fair or biased. Initially, the player believes there is an equal probability that the die is either fair or biased, with prior probabilities:

$$P(F) = 0.5, \quad P(B) = 0.5. \tag{A.30}$$

If the die is fair, each face appears with probability $1/6$, while if the die is biased, the probability of rolling a six is higher, say $P(6|B) = 0.3$, while other numbers are distributed accordingly. Suppose the player rolls the die five times and observes three sixes. Using Bayes' theorem, we update the probability that the die is biased given the observed rolls:

$$P(\theta|x) = \frac{P(x|\theta)P(\theta)}{\sum_\theta P(x|\theta)P(\theta)}. \tag{A.31}$$

where θ represents the hypothesis (fair or loaded) and x represents the observed data. The likelihood of rolling three sixes in five rolls under each scenario is computed as:

$$P(x|\theta = F) = \binom{5}{3}\left(\frac{1}{6}\right)^3\left(\frac{5}{6}\right)^2 = 0.032, \tag{A.32}$$

$$P(x|\theta = B) = \binom{5}{3}(0.3)^3(0.7)^2 = 0.13 \tag{A.33}$$

The $P(\theta)$ represents the prior belief that the player has, assume the player beliefs the chance to be fair or biased are %80 and %20 respectively, because the player thinks that the number of trials were not enough to claim that the dice is biased. Then,

$$P(\theta|x) = \begin{cases} \frac{0.032\times0.8}{(0.032\times0.8+0.132\times0.2)} = 0.492 & \text{if } \theta = F \\ \frac{0.132\times0.2}{(0.032\times0.8+0.132\times0.2)} = 0.507 & \text{if } \theta = B \end{cases} \tag{A.34}$$

1.3 Odds

Since we already defined probability, now we can introduce odds which can be defined as ratio between number of events that gives the outcome which we are looking for and ones that do not. Thus,

$$\text{Odds} = \frac{P(X)}{1 - P(X)} \tag{A.35}$$

and compare theoretical calculation of odds with a real one. Let's examine the odds of hitting even/odd pockets and straight up (single number e.g. 9) in a roulette round.

The pockets of the roulette wheel are numbered from 0 to 36 and odd numbers are red and even are black in ranges from from 1 to 10 and 19 to 28 and vice versa in ranges from 11 to 18 and 29 to 36. Also, there is green pocket marked as zero. Since both number of red pocket and number of black pocket are 18, probability of hitting even or odd number is equal as $P(x) = 18/37$. Moreover, since the probability of the ball occurring at 9 is $P(x) = 1/37$. Therefore, the odds of hitting even/odd and 9 are 18 to 36 and 1 to 36 respectively.

1.4 Moments of a Probability Distribution

Moments are quantitative measures used to describe the characteristics of a probability distribution.

1.4.1 Moments of discrete distributions

To characterize a probability distribution, we define its moments. The expected value, also known as the first moment, represents the average outcome of a random variable over a large number of trials.

$$\mu = \sum_{x_i} x_i P(x_i) \tag{A.36}$$

In the context of gambling, the expected value helps assess the average payout a player can expect over a large number of rounds. In roulette, a bet on a single number (straight bet) pays out 35 to 1 if successful. The probability of winning a straight-up bet is $1/37$, while the probability of losing is $36/37$. If a player bets 1\$ on a single pocket, they win 35\$ with probability $1/37$ and lose 1\$ with probability $36/37$. The expected value of this bet is calculated as follows:

$$\sum_{x_i} x_i P(x_i) = 35 \cdot \frac{1}{37} + (-1) \cdot \frac{36}{37} = -0.027 \tag{A.37}$$

This result means that, on average, the player loses approximately 0.027 for every dollar wagered on a single number over a large number of rounds.

The variance, also known as the second central moment, measures how the set of possible outcomes deviates from the expected value.

$$\begin{aligned} \sigma^2 &= \sum_{x_i} (x_i - \mu)^2 P(x_i) \\ &= \sum_{x_i} x_i^2 P(x_i) - \left(\sum_{x_i} x_i P(x_i) \right)^2 \end{aligned} \tag{A.38}$$

The standard deviation is defined as the square root of the variance:

$$\sigma = \sqrt{\sum_{x_i} (x_i^2 - \mu^2) P(x_i)} \tag{A.39}$$

For a single number bet in roulette, the variance is calculated as follows:

$$\begin{aligned}\sigma^2 &= \left[\left(35^2 \cdot \frac{1}{37} + (-1)^2 \cdot \frac{36}{37}\right) - \left(35 \cdot \frac{1}{37} + (-1) \cdot \frac{36}{37}\right)^2\right] \\ &= \left(\frac{1261}{37} - (-0.027)^2\right) \\ &= (34.08108 - 0.000729) = 34.08035\end{aligned} \tag{A.40}$$

The standard deviation is then:

$$\sigma = \sqrt{34.08035} = 5.83784 \tag{A.41}$$

1.4.2 Moments of continuous distributions

For continuous random variables, key statistical summaries such as the mean and standard deviation are defined similarly to the discrete case, but summations are replaced with integrals.

The mean, or expected value of X, is given by:

$$\mu = \int_{-\infty}^{\infty} x f(x) dx. \tag{A.42}$$

Similar to the discrete case, if one observes a large number of values of X, the sample mean $\bar{X}$ will approximate μ.

To measure the spread of values around the mean, we compute the variance:

$$\sigma^2 = \int_{-\infty}^{\infty} (x-\mu)^2 f(x) dx. \tag{A.43}$$

The standard deviation of X is defined as the square root of the variance.

1.5 Statistical Distributions

Under the topics of probability theory, we understand that a single experiment is not enough to estimate a system's behavior. In order to understand this behavior, a collection of N identical systems must be examined [27]. According to the type of outcome, which can be discrete or continuous, we have grouped distributions into two categories: discrete distributions for finite outcomes and continuous distributions for infinite outcomes. Examples of probability theory show that gambling games have finite outcomes; for instance, the experiment of dice rolling has 6 possible results. Thus, discrete distribution types, which include uniform, Bernoulli, binomial, and Poisson distributions, can be used for analysis. However, continuous distributions can also be used for examining losses and wins of a large number of rounds, such as normal distribution.

1.5.1 Discrete distributions

A discrete distribution describes the probability of outcomes for a discrete random variable, which takes on countable values (e.g., integers).

1.5.1.1 Uniform Distribution

A uniform distribution describes a system in which all possible outcomes are equally likely. For example, in a roulette game with 37 compartments (numbered 0 to 36), each compartment is equally likely due to the symmetry of the wheel. The probability of the ball landing in any specific compartment is given by

$$P(X = x_i) = \frac{1}{n}, \tag{A.44}$$

where n is the total number of outcomes.

1.5.1.2 Bernoulli Distribution

The Bernoulli distribution is used for experiments that have only two outcomes, often labeled as "success" and "failure." Let p be the probability of a success (e.g., obtaining heads in a coin toss) and $1-p$ the probability of a failure.

The probability mass function is defined as

$$P(X = x) = \begin{cases} p, & \text{if } x = 1 \quad \text{(success)} \\ 1-p, & \text{if } x = 0 \quad \text{(failure)} \end{cases} \tag{A.45}$$

The expected value is then calculated by

$$\langle x_i \rangle = 1 \cdot p + 0 \cdot (1-p) = p. \tag{A.46}$$

For example, if $p = 0.5$, the expected value is 0.5.

1.5.1.3 Binomial distribution

The binomial distribution models the probability of obtaining a specific number of successes in a fixed number of independent trials, each with the same probability p of success. Its probability mass function is

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}, \tag{A.47}$$

where n is the total number of trials and k is the number of successes.

For example, consider a slot machine with a jackpot feature that occurs with probability $p = 0.05$ per spin. If a player spins the machine 10 times, the probability of hitting the jackpot exactly 3 times is

$$\begin{aligned} P(X = 3) &= \binom{10}{3} (0.05)^3 (0.95)^7 \\ &\approx 0.01. \end{aligned} \tag{A.48}$$

The expected number of jackpots is

$$\langle x_i \rangle = n \cdot p = 10 \cdot 0.05 = 0.5, \tag{A.49}$$

and the variance is

$$\sigma^2 = n \cdot p \cdot (1-p) = 10 \cdot 0.05 \cdot 0.95 = 0.475. \tag{A.50}$$

1.5.2 Continuous distributions

A continuous distribution describes the probability of outcomes for a continuous random variable, which can take on any value within a range or interval.

1.5.2.1 Poisson Distribution

The Poisson distribution describes the probability of a given number of events occurring within a fixed interval, assuming the events occur with a constant mean rate λ and independently of each other. For instance, consider a slot game where a bonus feature occurs once every 100 spins, so that $\lambda = 0.01$ per spin. Over 50 spins, the average rate becomes

$$\lambda_{50} = \lambda \times 50 = 0.5. \tag{A.51}$$

The probability mass function is

$$P(X = k) = \frac{(\lambda_{50})^k e^{-\lambda_{50}}}{k!}. \tag{A.52}$$

For $k = 2$, the probability is calculated as

$$\begin{aligned} P(X = 2) &= \frac{(0.5)^2 e^{-0.5}}{2!} \\ &= \frac{0.25 \cdot e^{-0.5}}{2} \\ &\approxeq 0.076. \end{aligned} \tag{A.53}$$

Thus, the probability of hitting the bonus feature exactly twice in 50 spins is approximately 7.6%.

1.5.2.2 Normal (Gaussian) Distribution

The normal distribution is a continuous probability distribution characterized by its symmetric, bell-shaped curve. Its probability density function (PDF) is given by

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{A.54}$$

where μ is the mean and σ is the standard deviation.

For example, suppose a slot machine has an average winning amount of $\mu = \$4$ with a standard deviation of $\sigma = \$10$ per spin. The PDF at $x = 5$ is

$$\begin{aligned} f(5) &= \frac{1}{\sqrt{2\pi \cdot 10^2}} e^{-\frac{(5-4)^2}{2 \cdot 10^2}} \\ &= \frac{1}{\sqrt{2\pi \cdot 100}} e^{-\frac{1}{200}} \\ &\approx 0.0397. \end{aligned} \tag{A.55}$$

The cumulative distribution function (CDF) for the normal distribution is defined as

$$F(x) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right]. \tag{A.56}$$

Consider the total winnings from 100 spins, where the expected total winnings are

$$\mu_{\text{total}} = N\mu = 100 \cdot 4 = 400, \tag{A.57}$$

and the total standard deviation is

$$\sigma_{\text{total}} = \sqrt{N}\sigma = \sqrt{100} \cdot 10 = 100. \tag{A.58}$$

To determine the probability of total winnings exceeding \$500, we compute the Z-score:

$$Z = \frac{500 - 400}{100} = 1. \tag{A.59}$$

According to standard normal tables, this corresponds to an upper-tail probability of approximately 15.87%. Moreover, the probability that winnings fall between two values a and b is given by

$$P(a < X < b) = F(b) - F(a). \tag{A.60}$$

1.6 Bayesian Inference

We mentioned Bayes' Theorem in previous section 1.1.5, which utilizes a probability based on one's opinion and evidence. Similarly, Bayesian inference is a statistical framework that creates posterior belief (distribution) based on prior beliefs and new evidence (observed data) [28]. This iterative process enables continuous learning and decision making in uncertain environments. Hence, Bayesian inference is a method to make estimations about unknowns subjectively by using the values of those unknowns using statistical distributions such as the Beta distribution for probabilities and the Normal distribution for continuous variables [29].For inferring from an experiment using this method, it is assumed that data are independently and identically distributed (i.i.d.) from a single distribution with unknown parameters [30]. Bayesian method provides a systematic approach to making informed decisions in the face of uncertainty, ultimately improving competitive play in card games. This approach translates luck into skill and success. In card games, Bayesian inference provides a powerful tool for strategic decision making by continuously updating beliefs about hidden information [31]. We can combine current game information (observed data) with a player's belief based

on past experiences (prior) and create more powerful estimations about the actions taken by opponents especially actions like bluffing, and challenging. Modeling these events with a Beta distribution, which is ideal for representing probabilities in repeated scenarios [32].

Bayesian inference is a method of updating our beliefs about a probability parameter based on observed data. This is done using Bayes' theorem, which states:

$$p(\theta \mid x) = \frac{p(x \mid \theta)p(\theta)}{p(x)} \tag{A.61}$$

where $p(\theta \mid x)$ is the posterior distribution-our updated belief about θ after observing data x, $p(x \mid \theta)$ is the likelihood function-the probability of observing the data given a specific value of θ, $p(\theta)$ is the prior distribution-our initial belief about θ before seeing any data. $p(x)$ is the evidence (marginal likelihood)-a normalizing constant ensuring the posterior sums to 1.

1.6.1 Bayesian inference with beta distribution

Bayesian model can be described with beta distribution if the experiment can be explained with binomial approach which has four conditions: a fixed number of trials, only two outcomes, a fixed success probability, and independent trials [30].

To model the probability parameter θ (e.g., the probability of success in a Bernoulli trial), we use the Beta distribution as our prior:

$$p(\theta) = \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{B(\alpha,\beta)} \tag{A.62}$$

where α the prior count of successes plus one, β the prior count of failures plus one, and $B(\alpha,\beta)$ the Beta function, that is defined as:

$$B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \tag{A.63}$$

where $\Gamma(n)$ is the Gamma function, a continuous extension of the factorial function.

Suppose we observe n independent Bernoulli trials, with $\sum x = k$ successes. The likelihood function, representing the probability of obtaining the data given θ, follows a Binomial distribution:

$$p(x \mid \theta) = \binom{n}{k} \theta^k (1-\theta)^{n-k} \tag{A.64}$$

where $\binom{n}{k}$ is the binomial coefficient representing the number of ways to obtain k successes in n trials, θ^k the probability of obtaining exactly k successes and, $(1-\theta)^{n-k}$ the probability of obtaining $n-k$ failures.

Using Bayes' theorem, the posterior distribution is proportional to the likelihood times the prior:

$$p(\theta \mid x) \propto p(x \mid \theta) p(\theta) \tag{A.65}$$

Substituting the Binomial likelihood and the Beta prior:

$$p(\theta \mid x) \propto \theta^k (1-\theta)^{n-k} \times \theta^{\alpha-1} (1-\theta)^{\beta-1} \tag{A.66}$$

Simplifying the exponents:

$$p(\theta \mid x) \propto \theta^{(\alpha+k-1)} (1-\theta)^{(\beta+n-k-1)} \tag{A.67}$$

This is exactly the form of a Beta distribution:

$$p(\theta \mid x) = \text{Beta}(\alpha + k, \beta + (n-k)) \tag{A.68}$$

The posterior distribution ($k \to \sum x$)follows:

$$p(\theta|x) = \frac{\theta^{\alpha+\Sigma x-1}(1-\theta)^{\beta+n-\Sigma x-1}}{\int_0^1 \theta^{\alpha+\Sigma x-1}(1-\theta)^{\beta+n-\Sigma x-1} d\theta} \tag{A.69}$$

Thus, the prior counts of successes and failures are simply updated with the observed counts, leading to an intuitive and computationally efficient way of performing Bayesian inference.

For example, to estimate the probability of getting heads by flipping a coin: we assume an initial belief that the coin is fair but uncertain, setting a Beta(2,2) prior:

$$p(\theta) = \text{Beta}(2,2) \tag{A.70}$$

We flip the coin 10 times and observe 7 heads. Using the update rule we can substitute values below:

$$\theta \mid x \sim \text{Beta}(2+7, 2+(10-7)) = \text{Beta}(9,5) \tag{A.71}$$

1.6.2 Bayesian hierarchical modeling

Source of data comes from several subgroups which have their own characteristics (demographic differences, education level, individual differences etc.) in real-world problems. Classical approach examines one of two extremes: each subgroup separately or combines all data without considering the characteristics of those groups.

There are two approaches to handle this group estimation: separate estimates and combined estimates. Separate estimate approaches analyze each subgroup independently using its own data. This can arise a problem for small subgroups since the result can be unstable [30] (high variance). For example, if we denote the subgroup parameter by θ_i for group i, the estimation is carried out using the likelihood

$$y_i \sim p(y_i \mid \theta_i) \tag{A.72}$$

On the other hand, the combined estimates approach ignores the fact that there is a grouping variable and estimates the parameters in the combined sample. In this method, the subgroup parameters are assumed to come from a common population distribution. This is expressed as

$$\theta_i \sim p(\theta_i \mid \phi) \tag{A.73}$$

where ϕ represents the hyperparameters that capture the overall characteristics of the entire population. Although assuming that common population has the same characteristics allows us to obtain a more stable and low variance result in estimation, it provides a result that lacks consistency because we keep it independent of the subgroup features. For example, to diagnose a disease for a certain individual, the traditional Bayesian model uses prior data without acknowledging the individual's characteristics. However, in the real world, there are several conditions to make estimation more consistent, such as age, smoking, and patient history. If we consider characteristics of prior and current evidence, then our assumptions become more consistent.

Information from one group can inform the estimation in another, which is particularly valuable when some groups have only a few observations. The hierarchical model helps us to design this method: it creates inference largely based on overall information of the population when the subgroup has very little data, and if a subgroup has a large amount of data, then the estimation is driven largely by its own data, resembling a separate estimation. Hierarchical Bayesian modeling allows us to borrow strength across groups by extending the process to multiple levels.

$$\begin{aligned}
&\text{Level 1: Observations} \\
&\text{Data} \sim p(y \mid \theta)) \\
&\text{Level 2: Group parameters} \\
&\theta \sim p(\theta \mid \phi) \\
&\text{Level 3: Hyperparameters} \\
&\phi \sim p(\phi)
\end{aligned} \tag{A.74}$$

the variables in the equation below can be explained as follows: y is the observed data which outcomes collected from different groups, θ represents subgroup's characteristics and Φ defines the distribution that captures overall characteristics of the whole population which also defines local parameters. So, by extending the influence of the combined data with dependence on the available data for each subgroup, the hierarchical model creates a balance between all population and subgroup population data by combining them as below:

$$\phi_i, \theta_i | y_i \sim p(y_i \mid \theta_i) \cdot p(\theta_i \mid \phi) \cdot p(\phi) \tag{A.75}$$

Let α be the number of successes and β be the number of failures among n number of trials. Then, we can show success probabilities with Beta distribution.

$$\theta_i \mid \phi \sim \text{Beta}(\alpha, \beta), \tag{A.76}$$

with hyperparameters $\phi = (\alpha, \beta)$ governing the shape of the distribution. The Beta density is given by

$$p(\theta_i \mid \alpha, \beta) = \frac{\theta_i^{\alpha-1}(1-\theta_i)^{\beta-1}}{B(\alpha, \beta)}, \tag{A.77}$$

where $B(\alpha, \beta)$ is the beta function that ensures proper normalization. Since there is uncertainty about the hyperparameters as well, we assign them a hyperprior:

$$\phi \sim p(\phi). \tag{A.78}$$

Thus, the full hierarchical model combines these levels into

$$\phi_i, \theta_i \mid y_i \sim p(y_i \mid \theta_i) \cdot p(\theta_i \mid \phi) \cdot p(\phi). \tag{A.79}$$

This three-level model can be broken down as follows:

$$\begin{aligned} &\text{Level 1: Observations} \\ &\quad y_i \mid \theta_i \sim \text{Binomial}(n_i, \theta_i), \\ &\text{Level 2: Group Parameters} \\ &\quad \theta_i \mid \phi \sim \text{Beta}(\alpha, \beta), \\ &\text{Level 3: Hyperparameters} \\ &\quad \phi \sim p(\phi). \end{aligned} \tag{A.80}$$

In this setup, y_i represents the observed data (e.g., counts of successes), θ_i captures the characteristics of group i, and ϕ (with components α and β) describes the overall population features that, in turn, influence each subgroup.

An attractive feature of this hierarchical structure is its ability to "borrow strength" across groups. If a subgroup has few observations, its estimate for θ_i is informed by the overall population through ϕ. In contrast, when ample data exist for a subgroup, the estimation relies more on the subgroup's own data-essentially resembling a separate estimation.

Moreover, thanks to the conjugacy between the Beta prior and the Binomial likelihood, the posterior distribution for θ_i given y_i remains in the Beta family. Specifically, starting with the likelihood

$$p(y_i \mid \theta_i) = \binom{n_i}{y_i} \theta_i^{y_i}(1-\theta_i)^{n_i - y_i}, \tag{A.81}$$

and the prior

$$p(\theta_i \mid \alpha, \beta) = \frac{\theta_i^{\alpha-1}(1-\theta_i)^{\beta-1}}{B(\alpha, \beta)}, \tag{A.82}$$

the posterior becomes

$$p(\theta_i \mid y_i, \alpha, \beta) \propto \theta_i^{y_i+\alpha-1}(1-\theta_i)^{(n_i-y_i)+\beta-1}, \tag{A.83}$$

or equivalently,

$$\theta_i \mid y_i, \alpha, \beta \sim \text{Beta}(y_i+\alpha, n_i-y_i+\beta). \tag{A.84}$$

In conclusion, hierarchical Bayesian modeling with a Beta distribution allows us to balance subgroup-specific information with overall population trends if there are only two possible outcomes . By modeling

$$\phi_i, \theta_i \mid y_i \sim p(y_i \mid \theta_i) \cdot p(\theta_i \mid \phi) \cdot p(\phi), \tag{A.85}$$

we achieve a framework that leverages all available data by providing stable and consistent estimates while respecting the inherent differences among subgroups.

CURRICULUM VITAE

Name SURNAME: İrem ŞALK

EDUCATION:

- **B.Sc.:** 2022, Istanbul Technical University, Faculty of Science and Letter, Physics Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- May 2025 – Present: 8Great Games – Product Lead and Game Designer
- February 2024 – August 2024: Holago Games – Game and Level Designer
- May 2023 – February 2024: Zinky Games – Game and Level Designer
- April 2022 – May 2023: Mafia Games – Game Designer and Mathematician
- November 2021 – February 2022: TUBITAK-ITU – Undergraduate Research Scholar
- July – August 2020: ITU, Physics Engineering – Research Intern
- July – August 2019: TAEK, Technology Development Dept. – Training Intern
- September 2018 – July 2019: Istanbul Technical University – Undergraduate Assistant

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- **Salk I.**, Sarıel S. 2025: A Comparative Study for Analyzing LLM Performance in Bluff-Based Card Games. International Graduate Research Symposium – IGRS'2025, May 12–14, 2025, Istanbul, Turkey.