National College of Ireland

A Multi-Agentic AI System for Cricket Team Selection

MSc Research Project
Data Analytics

Ravi Sriraman
Student ID: 23411627

School of Computing
National College of Ireland

Supervisor: Jorge Basilo

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ravi Sriraman
Student ID:	23411627
Programme:	Data Analytics
Year:	2025
Module:	MSc Research Project
Supervisor:	Jorge Basilo
Submission Due Date:	11/12/2025
Project Title:	A Multi-Agentic AI System for Cricket Team Selection
Word Count:	6357
Page Count:	21

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	11th December 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

A Multi-Agentic AI System for Cricket Team Selection

Ravi Sriraman
23411627

Abstract

Data Analysis has become an integral part of any international or franchise cricket teams, almost every team have a dedicated data analyst with them. Data Analysts are required as finding insights from the historical data helps in picking players in auction, selecting players 11 for a match from the full squad, and to understand the strengths and weaknesses of both own team and the opponent team to plan how batting and bowling should be approached. However, the data analysts are really expensive, and have limitations as it is humanly really difficult to remember every statistics especially in a complex sport like cricket. This problem can be easily solved with the use of advancements in natural language processing techniques. We have built a multi-agent system with two flows: (1) Data analyst flow, and (2) team selector flow which helps in finding answers to any performance questions on any player, team, or venue, it also helps in finding the best playing 11 by picking the best performers in each role. The system is evaluated on three metrics: (1) Task Success rate, (2) Efficiency Score, (3) Correctness Score. The results show that the system achieves 100 percent task success rate and correctness score while scoring about 90 percent efficiency score.

1 Introduction

1.1 Background and Context

The application of data-driven decision-making has evolved significantly in most of the businesses and sports in last couple of decades. The "Moneyball" approach in baseball [1] demonstrated how data analysis can be used to discover the undervalued talent, challenging traditional scouting methods that primarily relied on expert's judgement. This paradigm shift has influenced multiple sports globally including cricket as it is well-suited with the availability of rich historical data.

Cricket team selection represents a complex multi-constraint optimization problem involving multiple interrelated factors: player role requirements (batters, bowlers, all-rounders, wicket-keepers), performance metrics (batting averages, strike rates, economy rates, wicket-taking ability), venue characteristics (pitch conditions, boundary sizes), opposition analysis (strengths, weaknesses), and tactical considerations (team balance, strategic flexibility, form fluctuations).

Traditional approaches to team selection rely heavily on expert judgement, historical knowledge, and subjective assessment of player form and fitness. While such expertise

remains valuable, it faces several inherent limitations. First, consistency across different selectors can vary significantly, leading to divergent team compositions for similar match situations. Second, cognitive biases—such as recency bias (overweighting recent performances), confirmation bias (seeking evidence that supports pre-existing beliefs), and availability bias (relying on memorable performances rather than consistent statistics)—can distort selection decisions.

We can use the advancements in Natural Language Processing and Agentic AI to make the decision-making process easier.

1.2 Problem Statement and Motivation

Selecting a playing 11 from a pool of 25 to 30 players is challenging. You want to understand the pitch you are going to play your next match on and pick the best players for each position. Other factors that impact the team selection are opposite team and their strengths and weaknesses.

Let us consider a situation where we want to select a playing 11 for RCB team against CSK and the match is going to be played in Chennai. To understand the pitch, we need answers to questions like "What is the average score at the Chennai pitch in last 3 years¿', "Which type of bowlers have been the most effective in this pitch¿', "Who are the top performers for CSK in Chennai¿'

Let us assume the answers to the above questions are: The average score is 170, spinners tend to take more wickets compared to the fast bowlers, R Gaikwad, RA Jadeja, A Mathre, and Pathirana are CSK's best performers. We can select players for RCB that have good record against CSK's best players, based on the average runs scored at this venue we can choose to select more bowlers or all-rounders, based on what kind of bowlers get more help from the pitch we can select suitable bowlers.

With current tools, the selectors are dependent on the Data Analysts to help them get answers to their questions and to select the final team. Writing queries to find answers to complex questions like "Who has the highest strike rate against X bowler between the overs 17 and 19 in Mumbai venue in last 3 seasons¿' is not too easy and if the Data Analysts make a simple mistake in the query, It can impact the outcome of the match.

This problem can be solved by simultaneously combining three critical elements that coaching staff require from team selection tools. First, natural language flexibility, allowing coaches to specify requirements conversationally using phrases like "Give me the names of top two bowlers who have taken V Kohli's wickets the most number times?" or "Select the best team for KKR team"—rather than requiring technical query languages or rigid API parameters. Second, interpretable decision-making providing transparent reasoning for each selection decision, such as "Player X was selected because their performance index is y"—enabling coaches to validate selections against their domain expertise and trust the system's recommendations. Third, dynamic constraint handling—adapting to varying tactical requirements across different match situations without requiring system reconfiguration or code changes. This research gap, where no existing system satisfactorily addresses all three requirements, motivates our investigation into multi-agent AI systems leveraging Large Language Models as a promising approach that could bridge this gap by combining natural language understanding (for flexibility), SQL query generation with visible logic (for interpretability), and stateful multi-agent orchestration (for dynamic constraint handling).

1.3 Research Question and Objectives

This study investigates: *Can a multi-agent AI system, leveraging Large Language Models and structured database queries, effectively automate cricket team selection while providing interpretable, constraint-satisfying solutions that adapt to natural language requirements?*

This research contributes:

Novel multi-agent architecture: A multi-agent system with sequential workflow gathers output from all the specialist agents and generate the final result.

A Cricket Analyst Agent: A chatbot that can write SQL queries automatically for the user's complex queries and find the right answers.

Cricket analytics database: An SQL database that stores data of 1,169 IPL matches (2008-2024), 800+ players, using four relational tables optimized for agent-based querying.

Evaluation framework: An evaluation framework that measures the Task Success Rate, Efficiency Score, and Correctness Score of both the team-selection and the cricket chatbot.

1.4 Paper Structure

Section 2 reviews related work in cricket analytics, multi-agent systems, and LLM applications, identifying research gaps. Section 3 describes data acquisition, preprocessing, database schema, and evaluation framework. Section 4 presents a novel multi-agent architecture, state management, workflow, and design rationale. Section 5 deals with implementation. Section 6 reports experimental results on task success rate, efficiency, and correction score. Section 7 summarizes findings, limitations, contributions, and future work.

2 Related Work

This section reviews literature in cricket analytics, multi-agent systems, and LLM-based applications, identifying gaps motivating this work.

2.1 Cricket Analytics and Team Selection

Application of analytics in sports started with "Moneyball" revolution [1]. Lemmer and Govan [2] introduced quantitative bowling measures that allowed team management to rank bowlers using a single metric that is calculated by combing multiple performance metrics such as economy, bowling average, and bowling strike rate. Athlete performance prediction system was introduced by Iyer and Sharda [3] using neural networks, achieving a decent match winner prediction accuracy of 68However, this system lacked interpretability and struggled with compositional constraints.

Swartz et al. [4] worked on optimizing batting order using simulation and dynamic programming. Though this method provided a solution to ordering of batsmen which has already been selected, it didn't deal with the selecting the team itself. Ahmed et al. [5] worked on optimizing the process of picking players in auction by leveraging NSGA-II algorithm. This approach treated team selection a s multi-objective optimization problem that tried to simultaneously maximize the batting and bowling performance

while respecting the budget constraints. Verma et al. [6] introduced similar but a modified algorith to deal with the same problem as Ahmed et al.

Fuzzy logic was also adopted by some researchers to model a team selection rule for cricket [7–9].

In the 2020's, machine learning models were adopted into the cricket team. Bunker and Thabtah [10] surveyed ML frameworks for sports prediction, highlighting accuracy-interpretability tradeoffs. Black-box models struggle with adoption due to lack of explainability.

2.2 Multi-Agent Systems and LLM-Based Agents

Though Multi-agent systems are more popular now, the multi-agent architectural patterns have existed since as early as 2009 introduced by Wooldridge [11]. One of the main feature that power these systems solve complex problems is the ability to decompose a large task into small independent tasks, which was identified by Veloso [12].

LLMs revitalized agent architectures. The architecture of the multi-agentic systems further evolved in the last 5 years integrating tools, LLMs, and structured output generation. Xi et al. [13] categorized LLM-based agents by autonomy and tool use, finding supervisor-based architectures excel at multistep tasks with global constraints. Wang et al. [14] contrasted monolithic and collaborative frameworks, showing multi-agent designs provide modularity, transparency, and fault tolerance.

LLMs are non-deterministic in nature, to ground these systems and to produce consistent results prompting techniques were developed. Wei et al. [15] introduced chain-of-thought prompting for improved reasoning. Later developers adopted ReAct patterns enabling interleaved tool use with reasoning which is critical for SQL agent workflows.

2.3 LLM Integration with Structured Data

Text-to-SQL gained attention with GPT-4 achieving 80%+ execution accuracy on Spider and WikiSQL. However, benchmarks focus on single queries; team selection requires sequential, stateful querying with progressive constraints.

LangChain [16] and LangGraph [17] provide frameworks for stateful LLM applications with tool binding and multi-agent orchestration.

2.4 Conclusion

Though there are multiple studies that address the cricket team selection, LLM based multi-agent system, and SQL agents, there aren't any work on combining all these technologies together. In this study, we have built multi-agent system leveraging, LLMs, LangGraph library, LangGraph's SQL tools, and custom tools built to calculate performance index, and to rank the players based on their role and performance.

2.5 Positioning of Current Work

This research addresses gaps by: (1) combining interpretability with optimization through transparent SQL generation, (2) enabling natural language interfaces eliminating technical query languages, (3) implementing stateful constraint management ensuring compositional correctness, and (4) providing first LLM-based multi-agent application to cricket team selection with empirical validation.

3 Research Methodology

A step-by-step procedure is required to conduct any research in data science field. This section describes every step involved in each phase from data collection to evaluating results. The Figure 1 illustrates the steps that can be followed to reproduce the research.

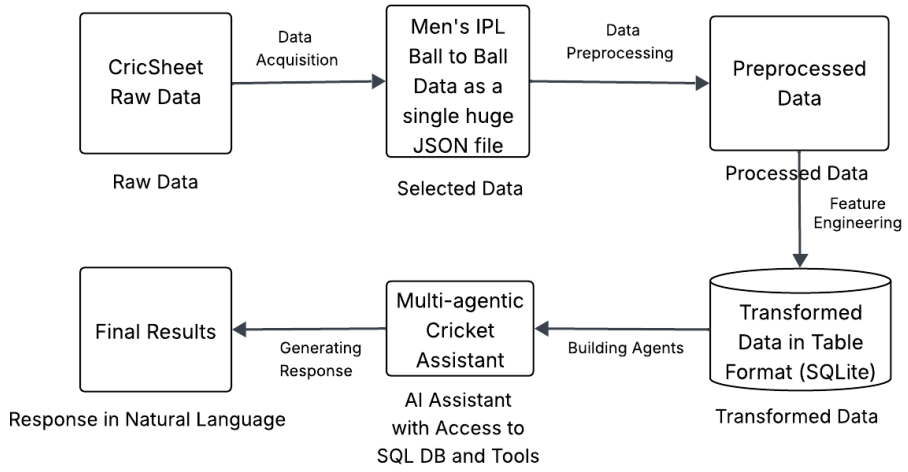

Figure 1: Data processing pipeline showing transformation of raw JSON match files to SQL tables.

3.1 Data Sources and Acquisition

A comprehensive cricket dataset is required to understand player's performance and to select a strong team based on multiple factors. There are a few sources where we can acquire this dataset from including cricbuzz, ESPN CricInfo, and CricSheet. While CricBuzz and ESPN CricInfo contains ball by ball data with commentary and player's statistics, it is private so we cannot use their data for research purposes. The only dataset that is available for free and allows to be used for research purpose is CricSheet. This is the reason we are making use of CricSheet dataset. Cricsheet dataset contains ball-ball-ball JSON files for 1,169 IPL matches from 2008 to 2024. Each file contains match metadata (venue, date, teams, toss, outcome) and delivery-level data (batter, bowler, runs breakdown, wickets).

3.2 Data Preprocessing

Raw JSON underwent multi-stage transformation to create structured relational tables optimized for query performance.

3.2.1 Team Name Normalization

As some of the franchises renamed their teams, the team names were not consistent. For example, "Delhi Daredevils" was rebranded to "Delhi Capitals". A comprehensive SQL normalization script (`update_team_names.sql`) was developed to standardize all team references to consistent abbreviations: CSK, SRH, LSG, RR, PBKS, RCB, GT, MI, DC, and KKR.

3.2.2 Venue Standardization

Venue names also exhibited significant variation across match files (e.g., "M Chinnaswamy Stadium" vs. "M.Chinnaswamy Stadium" vs. "M Chinnaswamy Stadium, Bengaluru"). A JSON mapping file (`stadiums.json`) was created to standardize 22 venue variants into canonical stadium names. For example, "M.Chinnaswamy Stadium" and "M Chinnaswamy Stadium, Bengaluru" were both mapped to "M Chinnaswamy Stadium". This mapping was applied during match data extraction to ensure consistent venue references across the database.

3.2.3 Match Metadata Extraction

Match metadata was extracted directly from JSON files: Match dates were preserved as strings in YYYY-MM-DD format from the `info.dates[0]` field. Team names were extracted from the `info.players.keys()` structure. Winner and outcome information (win by runs/wickets) were parsed from the `info.outcome` object. Match numbers and season information were extracted from `info.event` and `info.season` fields respectively. The night match classification was derived algorithmically for each match date, if no subsequent matches occurred on the same date with higher match numbers, the match was classified as a night match.

3.2.4 Ball-to-Ball Data Transformation

The nested JSON structure for delivery data required flattening into relational format while preserving all relevant information. Each delivery's nested structure was expanded into a flat record with explicit columns for each attribute. The attributes for each delivery included over number, delivery, batter, bowler, runs scored by batsman, runs conceded by bowler, batting team, bowling team, dismissed player, type of dismissal. Each delivery record retained match context and situational context to enable contextual analysis.

3.3 Database Design

SQLite database (`cricket.db`, 45 MB) comprises four tables:

`ipl_squads`: player_id, name, team, role, active (250+ records, indexed on team/role/name)

`matches`: match_id, date, venue, teams, toss, winner, margin, night_match, season (1,169 records, indexed on venue/date/season)

`ball_to_ball_data`: delivery_id, match_id, innings, over, ball, teams, players, runs breakdown, wickets (240,000 records, indexed on batter/bowler)

3.4 Dataset Statistical Analysis

To evaluate the quality of the dataset, We have performed a couple of statistical analysis: The first one is Temporal distribution which reveals the number of matches played every season and the second one is role distribution. Though have built an AI Assistant that helps in selecting the best players for each role, we need to configure the required number of players for each role. The analysis of role distribution helps in finding the common approach while deciding the number of players to select for each position in the team.

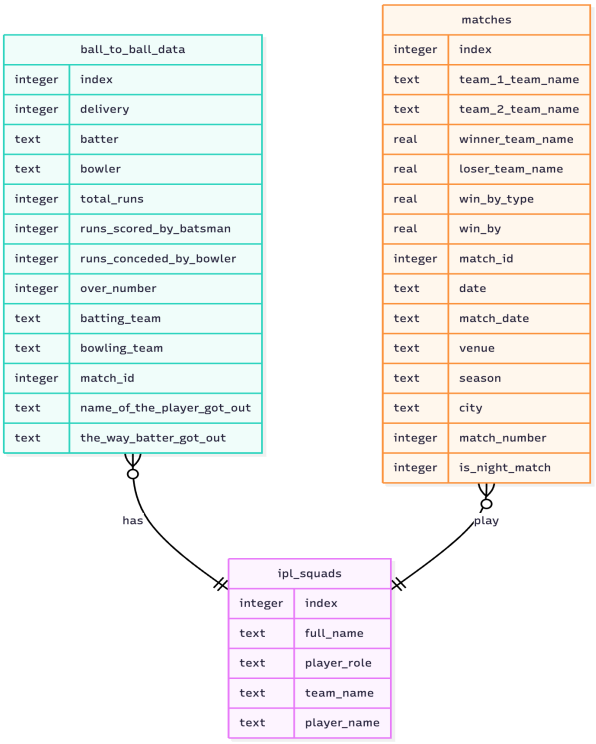

Figure 2: Entity-Relationship diagram showing the database schema structure with three main tables: ipl_squads, matches, and ball_to_ball_data, and their relationships.

3.4.1 Player Role Distribution

Figure 3 illustrates the distribution of player roles across current IPL team rosters. Teams average 7-8 specialist batters, 8-10 bowlers, 3-4 all-rounders, and 2-3 wicket-keepers, reflecting standard cricket squad structure. These numbers show the distribution of roles in their full squad (20-25 players), which directly influence the distribution of roles in the playing 11 as well helping us find the right configuration.

3.4.2 Temporal Coverage

Figure 4 demonstrates dataset temporal coverage spanning 17 IPL seasons (2008-2024). The number of matches played each year depends on the number of teams playing that year. In a tournament that has been running since 17 seasons, some teams go extinct while some new teams get added in the later years. 58-74 matches are played per year, with cumulative growth reaching 1,169 total matches. The rich dataset with the coverage of 1169 matches enable the assistant to answer various cricket questions and to select a team with better accuracy.

3.5 Building Multi-agent AI System

After performing data preprocessing and data transformation, the output was available in structured table format. We developed an SQL agent using an LLM and a well-crafted prompt that generates reliable SQL queries based on the request. A group of special agent nodes (one for each role) were designed that consists of an LLM, communicate with the SQL tools and ranking tools to select best players for each role. Finally, a team

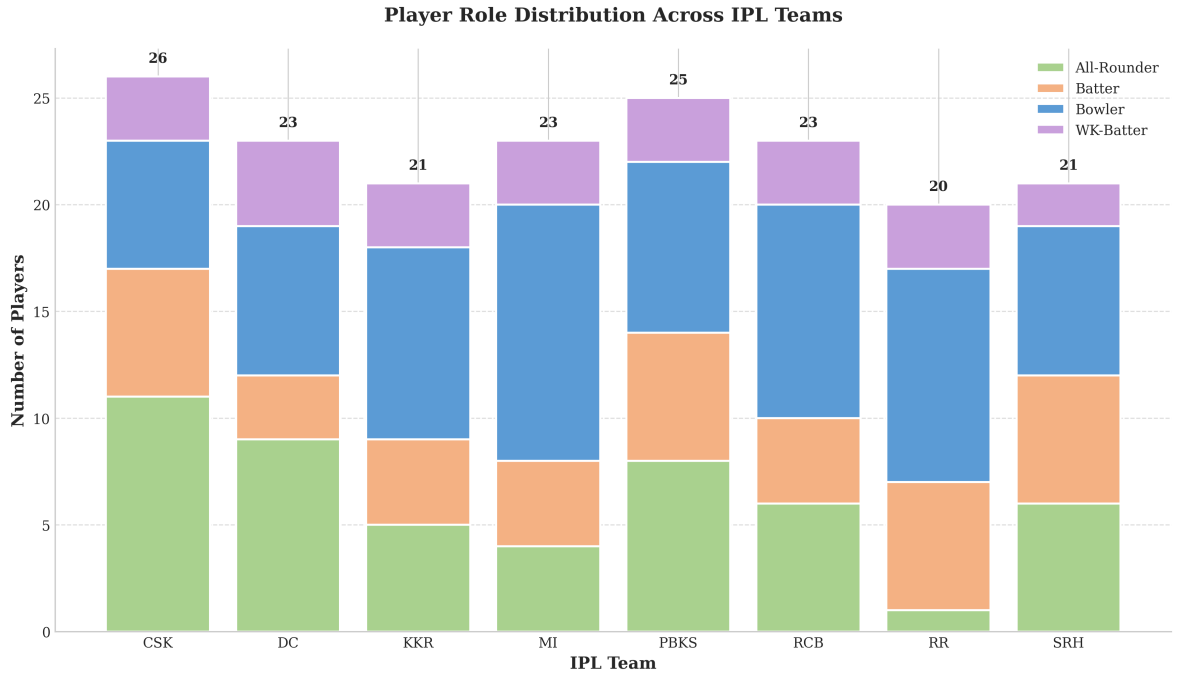

Figure 3: Player role distribution across 8 IPL teams showing balanced squad compositions.

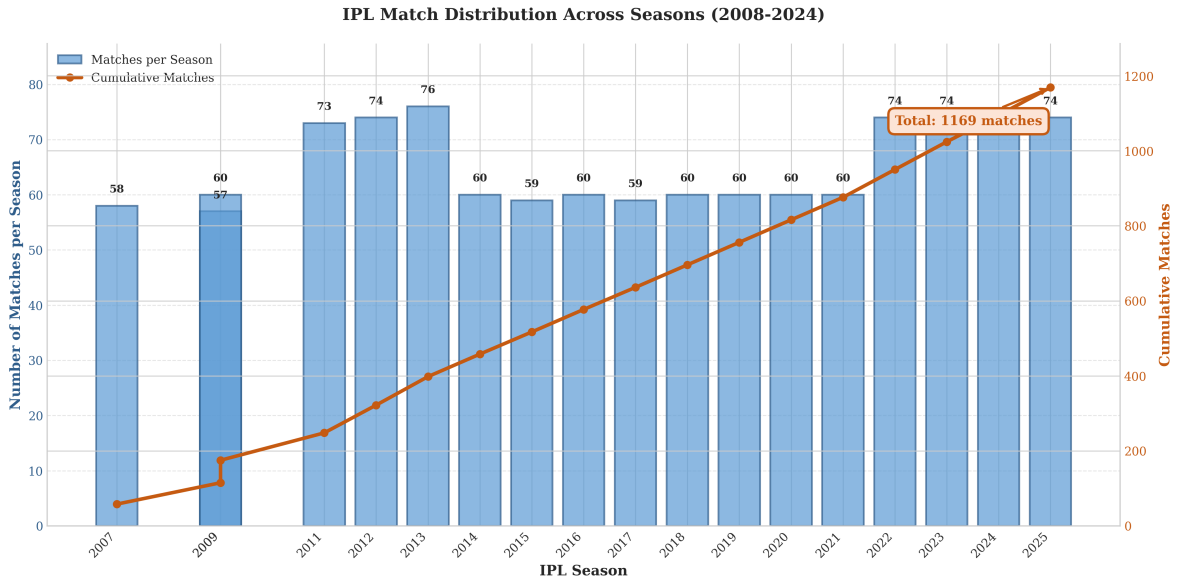

Figure 4: IPL match distribution across seasons (2008-2024) showing annual match counts (bars) and cumulative growth (line).

composition node was designed which helps to extract the best players for each role and come up with a final playing 11 and generate a response in the natural language.

3.6 Evaluation Methodology

As we have two flows in our system, we have developed two evaluation scripts to evaluate each flow. The field of evaluation of AI agent systems is still work in progress. How-

ever, the most widely used metrics to evaluate Agentic AI systems (Task Success Rate, Efficiency Score, and Correctness Score) [18] were used to evaluate the two flows in our system.

Task Success Rate: This metric measures how reliably the agent completes the task.

Efficiency Score: This metric measures assesses how efficiently an agent accomplishes the assigned task avoiding unnecessary steps and loops.

Correctness Score: This metric captures whether the agent's response was factually accurate.

As our assistant has two flows 5. We cannot calculate the evaluation metrics for both the flows using the same procedure. We have come up with a slightly different approach to calculate the evaluation metrics for the two flows. The procedure to calculate the these metrics are described below.

3.6.1 Evaluation of Cricket Analyst Flow:

Task Success Rate:

$$\text{Task Success Rate} = \frac{\text{passes\_checks}}{\text{total\_checks}} \times 100 \tag{1}$$

where `total_checks` includes: query_executed, generated_response, response_has_content, response_is_substantial, mentions_relevant_keywords

Efficiency Score:

$$\text{Efficiency Score} = (\text{step\_score} \times 0.4 + \text{sql\_score} \times 0.3 + \text{time\_score} \times 0.3) \tag{2}$$

where:

$$\text{step\_score} = \frac{\text{steps\_taken}}{\text{steps\_expected}} \tag{3}$$

$$\text{sql\_score} = \frac{\text{sql\_queries\_executed}}{\text{expected\_sql\_queries}} \tag{4}$$

$$\text{time\_score} = \begin{cases} 100 & \text{if execution\_time} < 10 \\ 100 - (\text{execution\_time} - 10) & \text{otherwise} \end{cases} \tag{5}$$

Correctness Score:

$$\text{correctness\_score} = \frac{\text{passed\_checks}}{\text{total\_checks}} \times 100 \tag{6}$$

where `total_checks` includes: has_answer, contains_data, has_numerical_results, mentions_context, no_error_messages, coherent_response

3.6.2 Evaluation of Team Selection Flow:

Task Success Rate:

$$\text{success\_rate} = \frac{\text{passed\_checks}}{\text{total\_checks}} \times 100 \tag{7}$$

where `total_checks` includes: workflow_completed, has_final_team, correct_team_size, has_wicket_keeper, has_batsmen, has_all_rounders, has_bowlers, correct_composition

Efficiency Score:

$$\text{efficiency\_score} = \begin{cases} \min\left(100, \frac{\text{expected\_steps}}{\text{step\_count}} \times 100\right) & \text{if step\_count} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

Correctness Score:

$$\text{correctness\_score} = \frac{\text{passed\_checks}}{\text{total\_checks}} \times 100 \tag{9}$$

where `total_checks` includes: players_have_names, players_have_metrics, wicket_keeper_valid, batsmen_valid, bowlers_valid, all_rounders_valid, performance_indices_present, no_duplicate_players

4 System Design

This section presents the multi-agent architecture, state management, and workflow.

4.1 System Architecture

We built a multi-agent system that answers any cricket stats question by computing the results using the data queried from cricket database, and it also helps in picking the best playing 11. Multi-agent systems are made up of different components that work together to accomplish a task.

When a user asks a general question like "What is Virat Kohli's batting strike rate between the overs 12 and 17? ". The query initially reaches the router node, the router node uses and LLM to decide whether to call team selection flow or cricket analysis flow. Since the above question is a generic question, langgraph runtime would route the flow to cricket_analyst.

If you want to find the best playing 11, you can ask "Pick a team for KKR." This time the router routes the request to the team selector flow as shown left side in the 5

The system employs a multi-agent system with sequential workflow [13] implemented using LangGraph [17]. Figure 5 provides an overview of the complete system architecture showing agent interactions, data flow.

The different nodes in the above system extract the required data from the agent state object, perform the task assigned to them using an LLM and tool calls, and finally, they save the results back to the agent state object.

4.1.1 Agent Nodes

The responsibilities of each agent node in our system are explained below.

Router: Routes the queries to the appropriate flow using an LLM, a conditional edge, and prompting [15] for selection workflow.

Specialist Agents (4): Four role-specific agents retrieve candidate pools:

- `wicket_keeper_agent`: Retrieves the best wicket keeper based on performance index.
- `batsmen_agent`: Retrieves all pure batsmen along with their performance metrics sorted by batting performance index

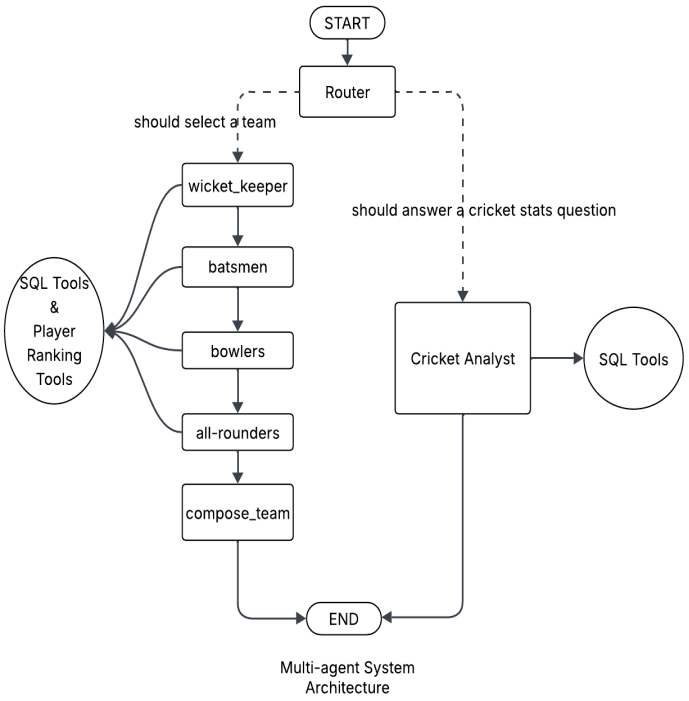

Figure 5: Multi-agent system architecture with 1 router, 4 specialized agents (wicket keeper, batsmen, bowlers all-rounders), 1 compose team agent for picking the best players and finalizing the playing 11, and 1 cricket analyst agent for answering data statistical questions, all interact with different tools

- `bowlers_agent`: Retrieves all pure bowlers along with their performance metrics sorted bowling performance index
- `all_rounders_agent`: Retrieves all all-rounders with their performance metrics sorted by their batting performance index and bowling performance index

compose_team: Extracts best players for each role from the candidates retrieved and ranked by the specialist agents to finally select the best team of 11 players.

4.1.2 State Management and Duplicate Prevention

State management is essential part of any multi-agent system, it enables the system to share output of one agent to the next agent so the agent can work on the task assigned to it.

The state object of our multi-agent system is made up of the following fields.

- **messages** field is used to store the conversation between the user and LLM (AI).
- **question** field is used to save the user's question.
- **answer** field is used to save the answer to the user's question in natural language by the cricket analyst if the query is related to generic cricket stats.
- **team_name** field stores the team name extracted from the user's question if the user requests to select a full team.

- **should_select_team** is a boolean field used by LangGraph runtime to decide whether to call cricket analyst flow or team selector flow.
- **wicket_keepers** field is used to save the best keeper name by the wicket_keeper node.
- **all_rounders** field is used by the all_rounders node to save the ranked performance metrics of all the all-rounders in the team.
- **batsmen** field is used by the batsmen node to save the ranked performance metrics of all the batsmen in the team.
- **bowlers** field is used by the bowlers node to save the ranked performance metrics of all the bowlers in the team.
- **final_team** field is used by the compose team node to save the selected team including 11 best performers.

4.1.3 Tool Calls:

Tool Calls enable agents to perform action than just generate a text response. The different tools that are used by our system are listed below. The SQL tools are from LangGraph [19] that makes it possible for the LLM to generate and valid SQL queries.

- **sql_db_schema:** Input to this tool is a comma-separated list of tables, output is the schema and sample rows for those tables. Be sure that the tables actually exist by calling sql_db_list_tables first! Example Input: table1, table2, table3.
- **sql_db_list_tables:** sql_db_list_tables: Input is an empty string, output is a comma-separated list of tables in the database.
- **sql_db_query:** Input to this tool is a detailed and correct SQL query, output is a result from the database. If the query is not correct, an error message will be returned. If an error is returned, rewrite the query, check the query, and try again. If you encounter an issue with Unknown column 'xxxx' in 'field list', use sql_db_schema to query the correct table fields.
- **sql_db_query_checker:** Use this tool to double check if your query is correct before executing it. Always use this tool before executing a query with sql_db_query!.
- **compute_batting_performance:** This tool computes BPF [6], given a player's batting strike rate and batting average.
- **compute_bowling_performance:** This tool computes CBR [6], given a player's bowling economy, bowling strike rate, and bowling average.
- **compute_all_round_performance:** This tool computes a player's all round performance by combing both batting performance and bowling performance.
- **rank_list_of_players_by_field:** This tool sorts a list of player's based on a selected field.

4.2 Agent Workflow

Based on the query type the router node routes the request to either Cricket Analyst Flow or Team Selector Flow 5. How both Workflows function is explained in detail below.

4.2.1 Cricket Analyst Flow:

Cricket analyst flow is made up of a single node called cricket_analyst_node.

cricket_analyst_node calls cricket analyst agent which uses few shot prompting technique and SQL tools receives a response in text format. The answer to the question is extracted using langchain's structured output extractor and cricket analyst agent has access to an LLM, a few shot prompting style system prompt, SQL tools such as sql_db_query, sql_db_schema, sql_db_list_tables, sql_db_query_checker, and LangGraph runtime.

4.2.2 Team Selector Flow:

Team selector node is built by combining 4 specialist agent nodes that help to fetch the performance metrics of each player and an aggregator called compose team node.

All these agents use an LLM, all SQL tools, tools to compute performance index, and tools to rank the players.

wicket_keeper node: this agent calculates strike_rate, average, performance index of each wicket_keeper. It also computes the BPF (batting performance factor) and saves the player statistics in both a JSON file and in the agent state.

batsmen node: batsmen node queries all batsmen and wicket keepers batsmen in the team, and computes batting_strike_rate, batting_average by writing and executing queries using SQL tools. It also calculates performance index by calling compute_batting_performance_index tool. Finally, it calls rank_batsmen tool to sort the batsmen based on their batting performance index BPF (batting performance factor) and then it saves player names along with their statistics in the agent_state object.

bowlers node: bowlers node selects all bowlers in the team, and computes bowling_strike_rate, bowling_average and bowling_economy. To compute these metrics bowlers agent writes and executes SQL queries using SQL tools. It also calculates performance index by calling compute_bowling_performance_index tool. Finally, it calls rank_bowlers tool to sort the bowlers based on their bowling performance index CBR (Combined bowling rate) and then it saves player names along with their statistics in the agent_state object.

all-rounders node: this node selects only the all-rounders in the team, and computes both batting and bowling metrics. To compute these metrics all-rounders agent writes and executes SQL queries using SQL tools. It also calculates performance index by calling compute_all_rounder_performance_index tool. Finally, it calls rank_all_rounders tool to sort the all_rounders based on all_rounder_performance index which is calculated by combining both BPF and CBR.

compose team node: compose team node does not use any LLM, it extracts the batsmen, bowlers, all_rounders, and wicket_keeper, and then it selects the n number top players where n is number of players configured for each role. In the end it updates the final_team value in the agent state object.

4.3 Architectural Advantages

We considered multiple options to make architectural decisions. The reasoning behind our architectural decisions are explained below.

Sequential Flow instead of Supervisor Flow: In our system we have used Sequential Flow because we don't want one special agent know about what the central agent performs or other agents have accomplished when a particular special agent is invoked. which is what happens with the supervisor architecture.

Context Engineering: Having as unpolluted and minimum context required for performing the task hand improves LLMs ability to generate better results. As we don't pass too much of context information from one node to another, The context remains very clean, so every node is able to call the right tools and generate the result in the desired format.

Optimized Token Usage: Since we are managing the context really well, we don't end up calling LLMs with large number of tokens which not only saves money but prevents possible token limit errors as well.

Extensibility: New agent types integrate without modifying existing agents.

5 Implementation

The AI assistant for cricket team selection was developed to assist team coaches in fetching complex player statistics and selecting 11 player team as shown in the figure 5. The system allows users to ask any question related to IPL matches by creating a plan, splitting the main task into multiple small tasks, writing SQL queries to complete small tasks, and finally combining the results of all small tasks to answer the user's query.

5.1 Implementation Environment

Agentic AI systems are powerful since they have a runtime, that calls LLMs to get instructions on what tools to call next and what arguments to send while calling those tools. Developing the runtime is time-consuming, and it is system that is needed for every agentic AI system, so we use a prebuilt runtime from a library. There are multiple python packages available that serve this purpose. However, we chose LangChain as it is an open-source and a stable library with a great community support.

Apart from our main library LangGraph, other software and hardware systems used while developing this system are as follows.

5.2 Technology Stack

Python 3.10: Extensive LLM libraries, database connectivity, type hints for maintainability.

LangChain 0.3: Unified API across LLM providers, prompt templates, tool/function calling, output parsing.

LangGraph 0.2: State graph representation, message routing, persistent state management, visualization for debugging.

GPT-4o: Strong SQL generation, complex reasoning, function calling. Temperature=0 for determinism.

SQLite 3.0: Lightweight, no separate server, full SQL support, ACID transactions, efficient indexing.

pandas 2.1: Statistical analysis, validation, CSV export.

Hardware: macOS 14, Apple M1, 16GB RAM

5.3 Tool Catalog

As discussed in the section 5.1, agents need tools to perform complex tasks. Tools that are required to list tables, read schema of the tables, and to validate the queries are provided by LangGraph. However, we had to build our own tools to perform some actions such as calculating batting performance index and bowling performance index.

Custom Tools:

- compute_batting_performance
- compute_bowling_performance
- compute_all_round_performance
- rank_list_of_players_based_on_by_field

5.4 SQL Query Generation

SQL query generation is core functionality of our system. Few-shot prompt with the tools mention in the section 5.3 are used to generate well-structured and optimized queries.

5.5 Structured Output

AI agents heavily depend on structured output without structured output, they won't be able to call tools as tools receive arguments which should be provided by the agents while calling them. We use prompt to instruct the LLM to generate the response in JSON format.

Using the same procedure the below agents extract data and add them to global agent state.

Router: updates should_select_team and team_name field **Cricket Analyst:** updates answer field
Wicket Keeper: updates wicket_keepers field
Batsmen: updates batsmen field
Bowlers: updates bowlers field
AllRounders: updates all_rounders field
Compose Team: updates final_team field

Once all nodes are executed, the driver program extracts and returns the value of either the final_team field or answer field from the agent_state.

5.6 Multi-agent Implementation

The implementation of our multi-agent system using python and Langgraph is shows in the 6

```
workflow = StateGraph(AgentState)
# Add nodes
workflow.add_node("router", router_node)
workflow.add_node("wicket_keeper", wicket_keeper_node)
workflow.add_node("all_rounders", all_rounders_node)
workflow.add_node("bowlers", bowlers_node)
workflow.add_node("batsmen", batsmen_node)
workflow.add_node("compose_team", compose_team_node)
workflow.add_node("cricket_analyst", cricket_analyst_node)
# Add edges - sequential workflow
workflow.add_edge(START, end_key: "router")
workflow.add_edge( start_key: "wicket_keeper", end_key: "batsmen")
workflow.add_edge( start_key: "batsmen", end_key: "bowlers")
workflow.add_edge( start_key: "bowlers", end_key: "all_rounders")
workflow.add_edge( start_key: "all_rounders", end_key: "compose_team")
workflow.add_edge( start_key: "compose_team", END)
workflow.add_edge( start_key: "cricket_analyst", END)
workflow.add_conditional_edges(
    source: "router", should_should_team, path_map: {"True": "wicket_keeper", "False": "cricket_analyst"}
)
```

Figure 6: Multi-agent system implementation

LangGraph library [17] offers a class called StateGraph which takes the agent state class as constructor parameter and initializes a graph structure. The agent state class that we define includes field names with type hints. These fields are accessed and updated by the agent nodes. The first node we have added to our graph is router node, followed by other specialist nodes. After adding all nodes, it is important to define the flow of execution which is achieved by connecting the nodes by edges. The edges are of two types: normal edges and conditional edges. The START node in our workflow is always connected to router node that uses the conditional edge node added in the last line of code as shown in the figure 6. Compose team and cricket analyst nodes are connected to the END node as the workflow exits once the execution of either of these nodes is over. Graphical representation of this workflow is shown in the figure 5.

6 Evaluation

6.1 Experiment 1: Data Analyst Flow

This subsection presents quantitative metrics from the comprehensive evaluation of the multi-agent cricket team selection system, including efficiency, correctness, task success rates, and cricket analyst performance. Below are the queries used to evaluate the data analyst flow of the agent and the answers returned by the agent.

What is V Kohli's batting average and strike rate?
"$batting\_average$" : 39.60, "$strike\_rate$" : 129.37

What is JJ Bumrah's bowling average and economy rate
"$bowling\_average$" : 20, "$economy\_rate$" : 7.03

Who has scored the most runs for CSK during 2024 and 2025?
The top scorer for CSK is S Dube, with a total of 753 runs

What is MS Dhoni's batting average against RCB?
39.39

Rank CSK batsmen based on their strike rate
["$batter$" : "$JOverton$", "$strike\_rate$" : 214.29, "$batter$" : "$UrvilPatel$", "$strike\_rate$" : 200.00, . . .]

Compare the batting averages of RG Sharma and V Kohli in IPL 2025
"RG_Sharma" : 29.86, "V_Kohli" : 54.75

Table 1 presents the cricket analyst agent's performance across diverse query types spanning simple statistics retrieval, team analysis, head-to-head comparisons, ranking tasks, and complex multistep queries. The agent achieved 100% task success and correctness across all six test cases, with efficiency scores ranging from 87.0% to 100% (average: 97.3%). The lowest efficiency (87.0%) for ranking queries reflects additional computational steps for sorting and ranking multiple players, while simpler statistical queries achieved near-perfect efficiency.

Test Category	Query Type	Success	Efficiency	Correct
Batting Stats	Player averages/SR	100%	98.3%	100%
Bowling Stats	Economy/averages	100%	100%	100%
Team Analysis	Top scorers	100%	98.6%	100%
Head-to-Head	Player vs team	100%	100%	100%
Ranking	Strike rate ranking	100%	87.0%	100%
Multi-Step	Complex comparisons	100%	100%	100%

Table 1: Cricket analyst agent performance across query categories

6.2 Experiment 2: Team Selector Flow

The team selection flow follows a different route in our agent architecture 5, compared to the data analyst flow. The tables 2, 3, and 4 show the efficiency score, task success rate, and correctness score of the team selection flow.

6.2.1 Team Selection Efficiency

The figure 7 shows the sample answer asked the agent to evaluate the team selector flow.
Query: "Select the best playing 11 for the CSK team."

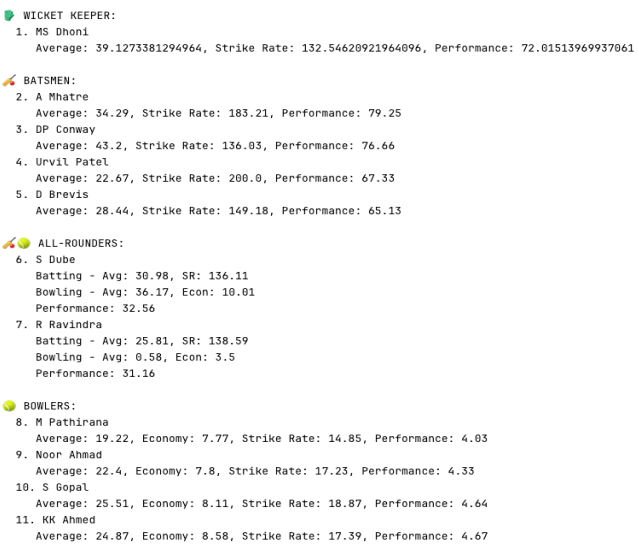

Figure 7: Team Selector Flow Sample Response.

Table 2 demonstrates that the system achieved optimal efficiency in completing the team selection workflow, executing the expected number of steps with 100% efficiency.

Metric	Expected	Actual	Efficiency Score
Steps	6	6	100%

Table 2: Team selection workflow efficiency metrics

6.2.2 Team Selection Correctness

Expected Condition	Met Condition
Players Have Names	Y
Players Have Metrics	Y
Wicket Keeper Valid	Y
Batsmen Valid	Y
Bowlers Valid	Y
All Rounders Valid	Y
Performance Indices Present	Y
No Duplicate Players	Y

Table 3: Team selection correctness validation results

Table 3 validates that all correctness conditions were satisfied, including proper player selection across all roles, presence of performance indices for ranking, and critical duplicate prevention.

6.2.3 Task Success Rate

Task	Is Successful
Workflow Completed	Y
Has Final Team	Y
Correct Team Size	Y
Has Wicket Keeper	Y
Has Batsmen	Y
Has All Rounders	Y
Has Bowlers	Y
Correct Composition	Y

Table 4: Task completion and composition validation results

Table 4 confirms 100% task success rate across all evaluation criteria, validating that the workflow produced a complete team with correct size and balanced composition across all required roles.

6.3 Discussion

While the efficiency score of the data analyst flow ranges between 87% and 100% depending on the complexity of the query, The system achieved 100% task success rate and correctness score. In case of team selector flow, the system achieved 100% task success rate, efficiency score, and correctness score due to its effective use of multiple specialist agents to perform each task.

7 Conclusion and Future Work

7.1 Research Summary

We built, tested, and evaluated a multi-agent assistant system for cricket team selection. The evaluation metrics show that it is possible to build a system leveraging moder LLMs, LangGraph agent runtime, SQL tools, and other custom tools, we can achieve 100 percent task completion accuracy, efficiency score, correctness score. The system is able to answer not only simple questions, but it is able to split a complicated query into small tasks, and able to accomplish them. Additionally, it is able to select the best players for each role by writing and executing queries, and ranking them by calling compute performance index, and ranking tools. Since we are using a sequential flow and saving all the intermediate results in a global agent state, we are not bombarding LLMs with too much of context data which could often potentially produce undesired results.

7.2 Key Findings

Your agent's results are only as good as the clarity in your prompt: Though LLMs are powerful at generating text and reasoning, they struggle when a task is complex and involves multiple steps. Prompting techniques like few shot learning ensures LLM plans tasks in a deterministic way and produce results that are consistent.

Multi-agent architecture is better than a single agent: When the task is complex and involves multiple decision-making, and computing multiple intermediate results to produce the final result, a single agent won't work. We need to build specialist agents to perform each unique task, save the results of each special agent in a global state, and then the intermediate results can be used by an aggregator to produce the final answer to the user's query.

Context Engineering is not a choice: LLMs struggle to produce consistent and correct results when the context is too large so it is crucial to keep only the essential part of the context than keeping every intermediate result or response.

Speed vs Performance: When employing LLM based agents to write SQL queries and fetch data, we need to compromise the speed for the flexibility offered by the natural language processing which is evident from our evaluation metrics.

7.3 Limitations

Though our system achieved excellent performance metrics, it comes with few limitations listed below.

Contextual factors: Though the cricket analyst agent answers venue-specific questions, or matchups, or recent form, etc. The cricket team selection flow considers only the player's performance in their role.

Team balance optimization: We have considered only the generic roles such as batters, bowlers, all-rounders, and wicket-keepers. However, there are other aspects to the sport such as how many spin bowlers are required or whether the opponent batsmen are strong against spin-bowling?.

7.4 Future Work

As cricket is a complex sport, it is not possible to build a solution that considers every aspect of the game at once. Though there are some limitations, this study can be used as a reference and the agent model can be extended to address all the limitations. Same architecture can be used to select teams for other sports including football, kabaddi, etc. The current system uses only IPL statistics. In the future, data from international matches, and domestic matches can also be integrated. We can also extend our model to support much complex decisions such as number of spin bowlers and number of fast bowlers required.

References

[1] M. Lewis, *Moneyball: The Art of Winning an Unfair Game.* W. W. Norton & Company, 2004.

[2] H. H. Lemmer, "The combined bowling rate as a measure of bowling performance in cricket," *South African Statistical Journal*, vol. 36, no. 2, pp. 103–117, 2002.

[3] S. R. Iyer and R. Sharda, "Prediction of athletes performance using neural networks: An application in cricket team selection," *Expert Systems with Applications*, vol. 36, no. 3, pp. 5510–5522, 2008.

[4] T. B. Swartz *et al.*, "Optimal batting orders in one-day cricket," *Computers & Operations Research*, vol. 33, no. 7, pp. 1939–1950, 2006.

[5] F. Ahmed, K. Passi *et al.*, "Player selection in cricket using data mining techniques," *International Journal of Computer Science Issues*, vol. 10, no. 2, pp. 1–8, 2013.

[6] S. Verma, V. Pandey, M. Pant, and V. Snasel, "A balanced squad for indian premier league using modified nsga-ii," *IEEE Access*, vol. 10, pp. 100 463–100 477, 2022.

[7] G. Singh, N. Bhatia, and S. Singh, "Fuzzy logic based cricket player performance evaluator," *International Journal of Computer Applications (Special Issue on Artificial Intelligence Techniques)*, pp. 11–16, 2011.

[8] C. D. Prakash and S. Verma, "A new in-form and role-based deep player performance index for player evaluation in t20 cricket," *Decision Analytics Journal*, vol. 2, p. 100025, 2022.

[9] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 7th ed. Pearson, 2015.

[10] R. P. Bunker and F. Thabtah, "A machine learning framework for sport result prediction," *Applied Computing and Informatics*, vol. 15, no. 1, pp. 27–33, 2019.

[11] M. Wooldridge, "An introduction to multiagent systems," 2009.

[12] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," in *Autonomous Robots*, vol. 8, no. 3, 2000, pp. 345–383.

[13] Z. Xi, W. Chen, X. Guo *et al.*, "The rise and potential of large language model based agents: A survey," *arXiv preprint arXiv:2309.07864*, 2023.

[14] L. Wang, C. Ma, X. Feng *et al.*, "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, 2024.

[15] J. Wei, X. Wang, D. Schuurmans *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[16] H. Chase, "Langchain: Building applications with llms," https://github.com/langchain-ai/langchain, 2023.

[17] L. AI, "Langgraph: Building stateful multi-actor applications with llms," https://github.com/langchain-ai/langgraph, 2024.

[18] T. College, "Evaluating ai agents in 2025: A practical guide," https://www.turingcollege.com/blog/evaluating-ai-agents-practical-guide, September 2025, accessed: 2025-12-02.

[19] L. AI, "Sql agent - langgraph documentation," https://docs.langchain.com/oss/python/langgraph/sql-agent, 2024, langGraph SQL Agent implementation guide.