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

National College of Ireland

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:	636
Page Count:	13

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

1 Initial Environment Setting

This section describes the software specifications and environment setup required for running the Multi-Agent Cricket Team Selection System.

1.1 System Setup Overview

Set up an Ubuntu-based virtual machine either on a local host or on any cloud provider Amazon Web Services (n.d.). Clone the project repository and follow either the automated one-command setup or the manual step-by-step setup described below.

1.2 Clone the Repository

```
git clone https://github.com/Ravi-Sriraman/agentic-ai-for-cricket-team-selection.git
cd agentic-ai-for-cricket-team-selection
```

1.3 Quick Start (Automated Setup)

If you want to set up everything required with a single command, export the API key and run the setup script:

```
export OPENAI_API_KEY=<Fetch from https://platform.openai.com/settings/organization/a
or request via x23411627@student.ncirl.ie>
chmod 777 ./setup.sh
sudo ./setup.sh
```

1.4 Rerun Workflow (Cleanup + Setup)

To re-run a clean setup, first run cleanup and then setup again:

```
chmod 777 ./cleanup.sh
sudo ./cleanup.sh
sudo ./setup.sh
```

1.5 Software Specifications

The software specifications for the setup are described in Table 1.

Component	Specification
Programming Language	Python (v3.10+)
AI Framework	LangChain, LangGraph
LLM Provider	OpenAI (GPT-4)
Database	SQLite 3
IDE	VS Code, PyCharm, Jupyter Lab
Version Control	Git

Table 1: Software Specifications

1.6 Environment Setup

Use the following instructions to set up the tools required for this project. You can either rely on the automated setup above or follow these manual steps:

1. **Set environment variables** (example values shown):

```
export DATABASE_URL=sqlite:///db/cricket.db
export OPENAI_API_KEY=<your_api_key>
export USER_AGENT=Ravi
```

2. **Install Ubuntu Packages Python Community (n.d.) GeeksforGeeks (n.d.)**:

```
sudo apt update -y
sudo apt install -y sqlite3
sudo apt install -y python3
sudo apt install -y python3-pip
sudo apt install -y python3-virtualenv
sudo apt install -y unzip
```

3. **Setup virtual environment and install python packages:**

```
virtualenv venv
source ./venv/bin/activate
./venv/bin/pip install -r requirements.txt
```

4. **Download dataset zip file from CricSheet:**

```
./venv/bin/python3 download_dataset.py
```

5. **Unzip the dataset file into data/ipl_json directory:**

```
unzip ./data/ipl_json.zip -d ./data/ipl_json
```

6. **Preprocess the data, create database table, and save data in database tables:**

```
./venv/bin/python3 setup_data.py
sqlite3 db/cricket.db < update_team_names.sql
```

1.7 Run and Test the System

Follow the instructions below to test the whole system or individual components

Evaluate Cricket Analyst Flow:

```
./venv/bin/python3 evaluate_cricket_analyst.py
```

Evaluate Team Selector Flow:

```
./venv/bin/python3 evaluate_team_selector.py
```

Test the System:

```
./venv/bin/python3 team_selector_workflow.py
```

Modify the question variable in the main function of team_selector_workflow.py to get answer to your analytical question

```
def main():
    load_dotenv()
    agent = create_supervisor_agent()
    #question = "Pick a playing 11 for PBKS from current squad based on the perfo
    question = "What is MS Dhoni's batting batting average against SP Narine?"

    # team_name = extract_team_name(question)

    initial_state = {
        "messages": [HumanMessage(content=question)],
        "answer": "",
        "question": question,
        "team_name": "",
        "wicket_keepers": [],
        "all_rounders": [],
        "bowlers": [],
        "batsmen": [],
        "final_team": {},
        "should_select_team": False,
    }

    print(f"\n{'='*80}")
    print(f"CRICKET CHAT WORKFLOW")
    print(f"{'='*80}\n")

    # Run workflow
```

```
final_state = None
for step in agent.stream(initial_state, stream_mode="updates"):
    for node_name, node_update in step.items():
        print(f"[{node_name}]")
        if "messages" in node_update and node_update["messages"]:
            for msg in node_update["messages"]:
                print(f"  {msg.content}")

        # Capture final state
        if "final_team" in node_update and node_update.get("final_team"):
            final_state = node_update
        elif "answer" in node_update and node_update["answer"]:
            print(f"  {node_update['answer']}")

# Display final team
if final_state and "final_team" in final_state:
    print_final_team(final_state["final_team"])
```

2 Code Implementation Details

This section describes the data preparation pipeline, database schema, design and development of the cricket team selection system. LangChain Chase (2023) and LangGraph AI (2024) were used to build the agents.

2.1 Download Data

```
response = requests.get("https://cricsheet.org/downloads/ipl_json.zip")
if not os.path.exists("./data/ipl_json"):
    os.mkdir("data/ipl_json")
with open("./data/ipl_json.zip", "wb") as f:
    f.write(response.content)
```

2.2 Data Sources

The system uses the following data sources:

1. **Ball-by-Ball Data**: Contains detailed information about every delivery in IPL matches
2. **IPL 2025 Squads**: Current team compositions and player roles
3. **Match Information**: Details about matches, venues, and dates

2.3 Database Schema

The SQLite database consists of the following tables:

2.3.1 ipl_squads Table

Stores current squad information for IPL 2025 teams.

```
CREATE TABLE ipl_squads (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    full_name   TEXT,
    player_role TEXT,
    team_name   TEXT,
    player_name TEXT
);
```

2.3.2 ball_by_ball Table

Stores detailed ball-by-ball data for analysis.

```
CREATE TABLE ball_by_ball (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    delivery                  INTEGER,
    batter                    TEXT,
    bowler                    TEXT,
    total_runs                INTEGER,
    runs_scored_by_batsman    INTEGER,
    runs_conceded_by_bowler   INTEGER,
    over_number               INTEGER,
    batting_team              TEXT,
    bowling_team              TEXT,
    match_id                  INTEGER,
    name_of_the_player_got_out TEXT,
    the_way_batter_got_out    TEXT
);
```

2.3.3 matches Table

Contains match information and metadata.

```
CREATE TABLE matches (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    team_1_team_name TEXT,
    team_2_team_name TEXT,
    winner_team_name REAL,
    loser_team_name  REAL,
    win_by_type      REAL,
    win_by           REAL,
    match_id         INTEGER,
    date             TEXT,
    match_date       TEXT,
    venue            TEXT,
    season           TEXT,
    city             TEXT,
```

```
    match_number     INTEGER,
    is_night_match   INTEGER
);
```

2.4 Preprocess Data

The python script 2.1 downloads ball to ball data as zip file. The zip file contains 1169 files, one file per match. The below code snippet shows how we combine the data from all the files and store the processed data in the respective tables.

```
    create_matches_table.main()
    create_ball_to_ball_table.main()
    create_current_squad_table.main()
    save_matches_in_db.main()
    save_ball_to_ball_table_in_db.main()
    save_ipl_squads_in_db.main()
```

2.5 Build Multi-agentic AI system using LangGraph

To build a multi-agentic AI system, we need different components including LLM, tools, an agent state, nodes, and edges. The following sections show the implementation details of each node and the agent state class. The AgentState class, and the code implementation of all agents is available in the file team_selector_workflow.py

2.5.1 Define AgentState

```
class AgentState(TypedDict):
    messages: Annotated[Sequence, add]
    team_name: str
    question: str
    answer: str
    should_select_team: str
    wicket_keepers: List[Dict[str, Any]]
    all_rounders: List[Dict[str, Any]]
    bowlers: List[Dict[str, Any]]
    batsmen: List[Dict[str, Any]]
    final_team: Dict[str, Any]
```

2.5.2 Router Node

```
def router_node(state: AgentState):
    """routes the requests to the right node"""
    llm = ChatOpenAI(model="gpt-4o")
    # Augment the LLM with schema for structured output
    structured_llm = llm.with_structured_output(SearchQuery)

    # Invoke the augmented LLM
    result = structured_llm.invoke(state["question"])
    return {
```

```
    "messages": [AIMessage(content=f" We should {'select a team'
                if result.should_select_team == 'True' else
                'answer a cricket question'}")],
    "team_name": result.team_name,
    "should_select_team": result.should_select_team
}
```

2.5.3 Data Analyst Node

```
def cricket_analyst_node(state: AgentState):
"""Answers cricket statistics questions."""

question = state.get("question", "Unknown")
agent = create_cricket_analyst_agent()
result = agent.invoke({"messages": [{"role": "user", "content": f"{question},
                                    return the response in JSON format
                                    with one field result"}]})
content = result["messages"][-1].content
llm = get_structured_llm()
output = llm.with_structured_output(CricketAnalystResult)
     .invoke(f"extract only the result field from json output. {content}")
return {
    "messages": [AIMessage(content=f"Final answer is {output.result}")],
    "answer": output.result
}
```

2.5.4 Wicket Keeper Node

```
def wicket_keeper_node(state: AgentState):
"""Select wicket keeper and extract structured data."""
team_name = state.get("team_name", "Unknown")
agent, _, _, _ = get_agents()
query = f"Select best wicket keeper batsman in {team_name}
         team and return in JSON format
         with fields player_name, batting_strike_rate,
         batting_average, performance_index"

result = agent.invoke({"messages": [{"role": "user", "content": query}]})
content = result["messages"][-1].content

# Extract structured data using Pydantic
wicket_keepers = extract_structured_data(content, WicketKeeperList)

# Save to file
filename = f"{team_name}_wicket_keepers.json"
save_json_to_file(wicket_keepers, filename)

return {
    "messages": [AIMessage(content=f" Wicket keepers analyzed
```

```
            ({len(wicket_keepers)} found)")],
        "wicket_keepers": wicket_keepers
    }
```

2.5.5 Batsmen Node

```
def batsmen_node(state: AgentState):
    """Select batsmen and extract structured data."""
    team_name = state.get("team_name", "Unknown")
    _, _, _, agent = get_agents()
    query = f"Find all batsmen in {team_name} and their
            performance in JSON format with fields player_name,
            batting_strike_rate, batting_average, performance_index"

    result = agent.invoke({"messages": [{"role": "user", "content": query}]})
    content = result["messages"][-1].content

    # Extract structured data using Pydantic
    batsmen = extract_structured_data(content, BatsmanList)

    # Save to file
    filename = f"{team_name}_batsmen.json"
    save_json_to_file(batsmen, filename)

    return {
        "messages": [AIMessage(content=f"Batsmen
                      analyzed ({len(batsmen)} found)")],
        "batsmen": batsmen
    }
```

2.5.6 Bowlers Node

```
    def bowlers_node(state: AgentState):
    """Select bowlers and extract structured data."""
    team_name = state.get("team_name", "Unknown")
    _, _, agent, _ = get_agents()
    query = f"Find top 10 bowlers in {team_name} and
              return their performance metrics in JSON format
              with fields player_name,bowling_average, bowling_economy,
              bowling_strike_rate, performance_index"

    result = agent.invoke({"messages": [{"role": "user", "content": query}]})
    content = result["messages"][-1].content

    # Extract structured data using Pydantic
    bowlers = extract_structured_data(content, BowlerList)

    # Save to file
    filename = f"{team_name}_bowlers.json"
```

```
save_json_to_file(bowlers, filename)

return {
    "messages": [AIMessage(content=f" Bowlers analyzed ({len(bowlers)} found)")],
    "bowlers": bowlers
}
```

2.5.7 All-Rounders Node

```
def all_rounders_node(state: AgentState):
"""Select all-rounders and extract structured data."""
team_name = state.get("team_name", "Unknown")
_, agent, _, _ = get_agents()
query = f"Find all All-Rounders performance of
           {team_name} team in JSON format with fields player_name,
           batting_strike_rate, batting_average, bowling_average, bowling_economy
           bowling_strike_rate, performance_index"

result = agent.invoke({"messages": [{"role": "user", "content": query}]})
content = result["messages"][-1].content

# Extract structured data using Pydantic
all_rounders = extract_structured_data(content, AllRounderList)

# Save to file
filename = f"{team_name}_all_rounders.json"
save_json_to_file(all_rounders, filename)

return {
    "messages": [AIMessage(content=f" All-rounders
                 analyzed ({len(all_rounders)} found)")],
    "all_rounders": all_rounders
}
```

2.5.8 Compose Team Node

```
def compose_team_node(state: AgentState):
 """
 Compose final team using deterministic selection based on performance_index.
 Team composition: 1 WK + 4 Batsmen + 2 All-rounders + 4 Bowlers = 11 players
 """
 team_name = state.get("team_name", "Unknown")

 # Get data from state
 wicket_keepers = state.get("wicket_keepers", [])
 all_rounders = state.get("all_rounders", [])
 bowlers = state.get("bowlers", [])
 batsmen = state.get("batsmen", [])
```

```
# Sort by performance_index
wicket_keepers_sorted = sorted(
    wicket_keepers,
    key=lambda x: float(x.get("performance_index", 0)),
    reverse=True
)
all_rounders_sorted = sorted(
    all_rounders,
    key=lambda x: float(x.get("performance_index", 0)),
    reverse=True
)
bowlers_sorted = sorted(
    bowlers,
    key=lambda x: float(x.get("performance_index", 0)),
    reverse=False  # Lower is better for bowlers
)
batsmen_sorted = sorted(
    batsmen,
    key=lambda x: float(x.get("performance_index", 0)),
    reverse=True
)
not_wk = lambda batsman: batsman["player_name"] !=
              wicket_keepers_sorted[0]["player_name"], batsmen_sorted
batsmen_sorted = list(filter(not_wk))
# Select players deterministically
final_team = {
    "team_name": team_name,
    "wicket_keeper": wicket_keepers_sorted[0] if wicket_keepers_sorted else None,
    "batsmen": batsmen_sorted[:NUMBER_OF_BATSMEN],
    "all_rounders": all_rounders_sorted[:NUMBER_OF_ALL_ROUNDERS],
    "bowlers": bowlers_sorted[:NUMBER_OF_BOWLERS],
    "total_players": 11
}

# Save final team to file
filename = f"{team_name}_final_team.json"
filepath = RESULTS_DIR / filename
with open(filepath, 'w') as f:
    json.dump(final_team, f, indent=2)
print(f"  → Final team saved to {filepath}")

# Count selected players
selected_count = (
    (1 if final_team["wicket_keeper"] else 0) +
    len(final_team["batsmen"]) +
    len(final_team["all_rounders"]) +
    len(final_team["bowlers"])
)
```

```
    return {
        "messages": [AIMessage(content=f" Final team
                     composed ({selected_count} players)")],
        "final_team": final_team
    }
```

2.5.9 Building the Final Graph Using LangGraph

```
def create_supervisor_agent():
    """
    Create supervisor agent with structured output workflow.

    Features:
    - Uses Pydantic models with with_structured_output() for data extraction
    - No manual JSON parsing needed
    - Type-safe and validated data
    - Saves results to files
    - Deterministic selection by performance_index
    """
    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, "router")
    workflow.add_edge("wicket_keeper", "batsmen")
    workflow.add_edge("batsmen", "bowlers")
    workflow.add_edge("bowlers", "all_rounders")
    workflow.add_edge("all_rounders", "compose_team")
    workflow.add_edge("compose_team", END)
    workflow.add_edge("cricket_analyst", END)

    workflow.add_conditional_edges(
        "router", should_should_team, {"True": "wicket_keeper",
                                       "False": "cricket_analyst"}
    )

    return workflow.compile()
```

2.5.10 Invoke the agent with queries

```
ef main():
load_dotenv()
agent = create_supervisor_agent()
# question = "Pick a playing 11 for PBKS from
 current squad based on the performance metrics."
question = "Select the best playing 11 for CSK team?"

# team_name = extract_team_name(question)

initial_state = {
    "messages": [HumanMessage(content=question)],
    "answer": "",
    "question": question,
    "team_name": "",
    "wicket_keepers": [],
    "all_rounders": [],
    "bowlers": [],
    "batsmen": [],
    "final_team": {},
    "should_select_team": False,
}

print(f"\n{'='*80}")
print(f"CRICKET CHAT WORKFLOW")
print(f"{'='*80}\n")

# Run workflow
final_state = None
for step in agent.stream(initial_state, stream_mode="updates"):
    for node_name, node_update in step.items():
        print(f"[{node_name}]")
        if "messages" in node_update and node_update["messages"]:
            for msg in node_update["messages"]:
                print(f"  {msg.content}")

        # Capture final state
        if "final_team" in node_update and node_update.get("final_team"):
            final_state = node_update
        elif "answer" in node_update and node_update["answer"]:
            print(f"  {node_update['answer']}")

# Display final team
if final_state and "final_team" in final_state:
    print_final_team(final_state["final_team"])
```

References

AI, L. (2024). Langgraph: Building stateful multi-actor applications with llms, https://github.com/langchain-ai/langgraph.

Amazon Web Services (n.d.). Getting started with kinesis video streams on ubuntu, https://docs.aws.amazon.com/kinesisvideostreams/latest/dg/gs-ubuntu.html. Accessed: 2025-12-08.

Chase, H. (2023). Langchain: Building applications with llms, https://github.com/langchain-ai/langchain.

GeeksforGeeks (n.d.). How to install sqlite 3 in ubuntu, https://www.geeksforgeeks.org/installation-guide/how-to-install-sqlite-3-in-ubuntu/. Accessed: 2025-12-08.

Python Community (n.d.). Install python 3.11.9 on ubuntu, https://discuss.python.org/t/install-python-3-11-9-on-ubuntu/51093. Accessed: 2025-12-08.