CRICBOT

Shaik Omar Mukhtar (OXS230000), Pranavi Gayathri Somishetti (PXS230003)

The University of Texas at Dallas

CS 6320 Natural Language Processing

6th May 2025

Contents

Introduction

Cricket is more than a sport—it's a passion for millions. With the growing interest in real-time data and fan engagement, we developed **CRICBOT**, an AI-powered chatbot that assists cricket fans by answering queries about live matches, player statistics, game rules, trivia, and historical data. The bot aims to act as an intelligent companion for enthusiasts, offering both factual responses and conversational interaction using natural language processing.

System Overview

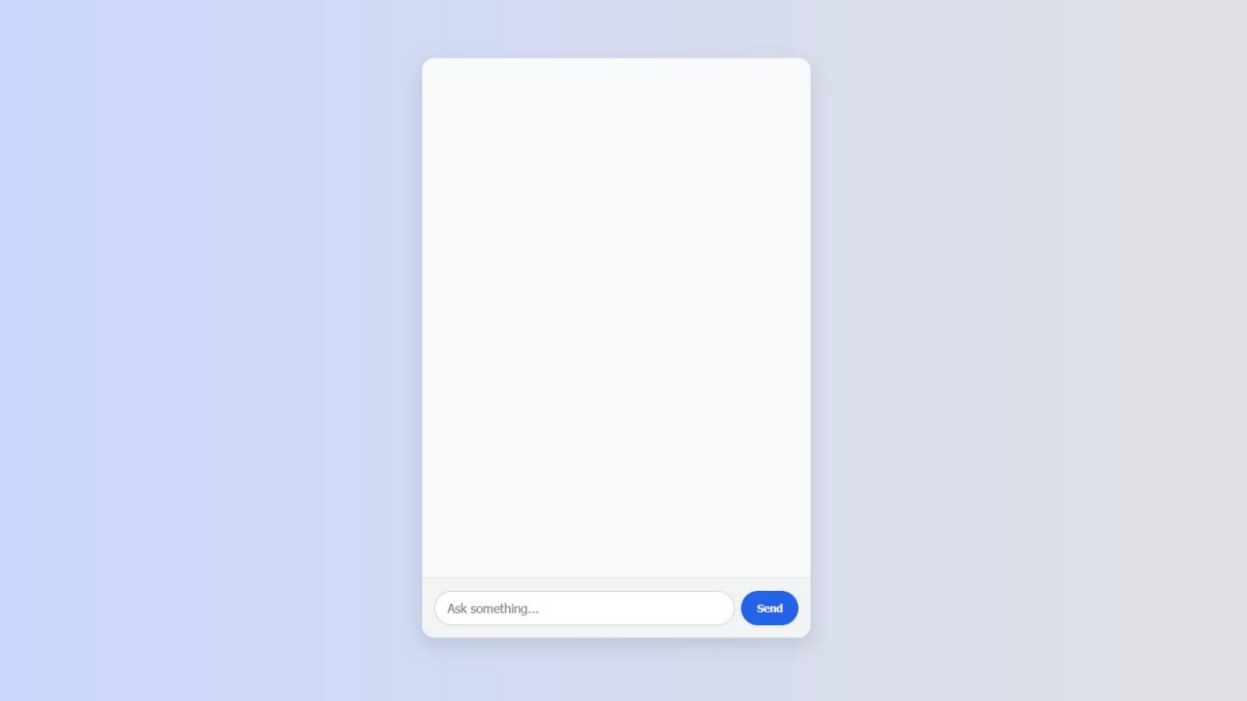
The CRICBOT architecture includes the following components:

1. Interface: The bot has a clean web-based interface, which accepts queries from the user.
2. Intent Recognition: Classifies the user's queries.
3. Named Entity Recognition: Identifies cricket-specific entities such as player names, tournaments, and scores.
4. Data Fetcher: Fetches actual data from various sources, uses API to websites such as cricAPI.

Technical Design

Interface

CRICBOT offers a clean, intuitive web-based interface designed to make cricket-related queries accessible and engaging for users of all levels. The front end is built using React, chosen for its component-based architecture and flexibility in building dynamic user interfaces. The backend APIs are developed using FastAPI, a high-performance Python web framework known for its speed and ease of integration with machine learning models. The screenshot below shows the UI.



Intent Recognition

The next step, after the user enters a query is to pre-process it and pass it through an intent recognition pipeline.

For intent recognition, we implemented a supervised learning pipeline using a pre-trained BERT model (bert-base-uncased) from the Hugging Face Transformers library. To fine tune the model, we created a dataset containing some questions and a classification label for each question. Listed below are a couple of examples.

Q: Has there been a high-scoring game at Eden Gardens recently? **Label:** match_information

Q: How many runs has Virat Kohli scored in ODIs? **Label:** player_information

The size of this dataset is about 370 questions, spanning 8 different classification labels. The model was trained for 8 epochs.

Named Entity Recognition

For this pipeline, we used Spacy library's ner pipeline. For the training dataset, initially we tried manually labelling different sentences we used in the Entity recognition pipeline, using Label Studio tool, but it proved to be a cumbersome task. We came up with a script to create a synthetic dataset and label it automatically and format it so that it can be used to train the Huggingface transformer model. We designed a specialized label set for the cricket domain, including player names, tournament names, team names etc. The script defines several static lists of domain-specific entities. It uses templated sentence structures (e.g., "In {year}, {player} scored 120 runs against {team} at {venue} in the {tournament}") where entity placeholders are dynamically replaced with randomly sampled values from the static lists. Each generated sentence is annotated using the BIO (Begin-Inside-Outside) format. (e.g., B-Player_Names, I-Tournament, etc.). The final output is a JSON file with 1500 labeled examples, structured in a format suitable for Spacy token classification tasks.

The Spacy NER model was trained using a transition-based architecture with a custom tok2vec embedding layer. The embedding component utilizes hashed features such as token shape, prefixes, suffixes, and normalized forms, processed through a multi-layer Maxout window encoder with a width of 96. The NER component receives these token vectors via a Tok2VecListener and applies a transition-based parser with a hidden layer width of 64. Training was performed from scratch without any pretrained vectors, using the Adam optimizer with weight decay, gradient clipping, and a low initial learning rate scheduled with linear warmup. The model was trained using a compounding batch strategy, evaluated every 200 steps, and guided by early stopping with a patience of 1600 steps. The training objective focused solely on maximizing the entity-level F1 score (ENTS_F). A maximum of 20,000 update steps was set, though training could halt earlier if performance plateaued. In our case it plateaued at ~3000 steps.

Database setup

To ensure accurate and reliable responses from the chatbot and avoid hallucinated outputs from language models, we set up a structured SQLite3 database using JSON files downloaded from Cricsheet (https://cricsheet.org/). A custom Python script was written to parse these JSON match files and extract relevant information such as player statistics, match summaries, team performances, and tournament details. This data was then organized into normalized SQL tables for player stats, match stats, team stats, and other relevant entities. By querying this database directly, the chatbot can provide precise, fact-based answers grounded in actual match data, significantly improving the trustworthiness and factual accuracy of responses related to cricket statistics.

Query generation

Based on the output obtained from intent recognition and entity recognition, we created some functions to translate these entities into actual database queries.

GPT backend

After obtaining query results from the database, these are passed to OpenAI's GPT API for text generation around this context. This backend also serves as a fallback, when the entity recognition or intent recognition pipelines fail, this helps answer the queries of the user. It can also answer any non-cricket specific queries.

Lessons learned

Working with unstructured data requires robust parsing and validation. Cricsheet JSON files contain complex nested structures. Extracting consistent stats across matches required a good understanding of the schema and careful handling of edge cases, such as missing fields or inconsistent player/team names.

NER training requires clean, diverse data and robust preprocessing. During the Spacy NER model training, we encountered issues like overlapping entities and span alignment errors. This taught us the importance of validating training data, using alignment modes, and augmenting datasets with varied, high-quality examples.

Evaluating model quality goes beyond accuracy metrics. Even though our NER model achieved a high F1 score during training, some test cases still failed. This showed us the value of manual testing with realistic inputs to catch generalization issues early.

Combining retrieval + generation improves chatbot quality. We learned that a hybrid approach — using SQL or vector search for retrieval and a language model for response generation — offers the best balance between factual accuracy and conversational fluency.

Contributions

Omar Mukthar Shaik – Created and annotated dataset for intent recognition pipeline, Trained, tested and finetuned the intent recognition model, Setup the web application and created the framework to integrate the models into the web application. Researched and tested other APIs available online (alternative to creating a local database), we decided not to use them as they are not suitable for our application.

Pranavi Gayathri Somishetti – Created synthetic annotated dataset for Named Entity Recognition, trained and tested the NER model. Experimented with hugging face bert transformer model and Spacy models for NER. Tried finetuning the models by changing the size of the datasets, changing the sentence templates, changing the training parameters like the learning rate and the number of steps. Spacy gave better results while taking less time to train. Set up the script to parse the JSON files and create the Database.

Testing outside the team and drawbacks identified

We deployed the application locally and asked a couple of our friends to try it out. It answered simple queries like “How does Virat Kohli perform?” pretty well. It struggles with more complicated queries like “Give me the statistics of player x, player y but not player z”. GPT is a fallback option, but it might hallucinate or give incomplete answers. Another drawback is that even with the local database, the application doesn’t always have access to the latest data. This requires implementing a RAG like model. This can be done in the future.

Links

Youtube

Github

Self-scoring

Metric	Pranavi Gayathri Somishetti (pxs230003)	Shaik Omar Mukhtar (oxs230000)
significant exploration beyond baseline	80	80
Innovation or Creativity	30	30
creative data augmentations	10	10
highlighted complexity	10	10
discussion of lessons learned and potential improvements	10	10
exceptional visualization/diagrams/repo	10	10

discussion of testing outside of the team, on 5 people	10	10
Total	**160**	**160**