

# **AI Course & Career Advisor Chatbot**

*A project report submitted to ICT Academy of Kerala  
in partial fulfillment of the requirements  
for the certification of*

## **CERTIFIED SPECIALIST IN DATA SCIENCE AND ANALYTICS**

submitted by

**SIBIN S S**

**HARIKRISHNAN R T**

**FIROZ MUHAMMED N**



**ICT ACADEMY OF KERALA  
THIRUVANANTHAPURAM, KERALA, INDIA  
Dec 2025**

## **List of Figures**

<b>SL.NO</b>	<b>TITLE</b>	<b>PAGE NO.</b>
FIG 5.1 (a)	HOMEPAGE	19
FIG 5.1 (b)	SUGGESTION BUTTONS	19

## List of Abbreviations

AI	ARTIFICIAL INTELLIGENCE
API	APPLICATION PROGRAMMING INTERFACE
CSV	COMMA -SEPERATED VALUES
CSS	CASCADING STYLE SHEETS
DIET	DUAL INTENT AND ENTITY TRANSFORMER
FAQ	FREQUENTLY ASKED QUESTIONS
ICTAK	ICT ACADEMY OF KERALA
ML	MACHINE LEARNING
NLU	NATURAL LANGUAGE UNDERSTANDING
NLP	NATURAL LANGUAGE PROCESSING
UX	USER EXPERIENCE

# **Table of Contents**

<b>ABSTRACT</b>	<b>1</b>
<b>1. PROBLEM STATEMENT</b>	<b>2</b>
1.1 OVERVIEW	2
1.2 PROBLEM STATEMENT	2
<b>2. INTRODUCTION</b>	<b>3</b>
<b>3. LITERATURE SURVEY</b>	<b>5-6</b>
<b>4. METHODOLOGY</b>	<b>7</b>
4.1 DATA COLLECTION	7
4.2 NATURAL LANGUAGE UNDERSTANDING(NLU)	7-9
4.3 RECOMMENDATION ALGORITHM	9-12
4.4 DIALOGUE MANAGEMENT	13-14
4.5 MODEL TRAINING	14-15
<b>5. WEB APPLICATION DEVELOPMENT</b>	<b>16</b>
5.1 SYSTEM ARCHITECTURE	16
<b>6. RESULTS</b>	<b>20</b>
<b>7. CONCLUSION</b>	<b>21</b>
<b>8. REFERENCES</b>	<b>22</b>

# Abstract

This project presents ICT Sahayi, an intelligent conversational chatbot designed to assist students in selecting appropriate IT certification courses from ICT Academy of Kerala's (ICTAK) offerings. The system addresses the challenge of information overload and decision paralysis faced by prospective students when choosing among multiple specialized courses. Using the Rasa open-source framework for natural language understanding (NLU) and dialogue management, combined with a custom-designed weighted scoring algorithm, the chatbot provides personalized course recommendations through an interactive four-question flow.

The system employs a sophisticated recommendation engine that analyzes user profiles across four key dimensions: career goals (10% weight), technical experience level (20% weight), area of interest (40% weight), and target career role (30% weight). This weighted approach ensures that recommendations are both accurate and contextually relevant. The chatbot achieves 95% recommendation accuracy and reduces the average course selection time from 30-45 minutes (manual counseling) to under 5 minutes, representing a 70-85% improvement in efficiency.

Key features include intelligent handling of uncertain users through discovery questions, progressive information disclosure to prevent cognitive overload, fuzzy string matching for error-tolerant course search, and context-aware suggestion buttons that guide users through their decision journey. The system is implemented using Python 3.8+, Rasa 3.6.13, Flask 2.x for the backend API, and vanilla JavaScript for the frontend interface. Course data is stored in CSV format for easy maintenance and updates.

Results demonstrate significant improvements in user engagement with an 85% completion rate for the advisor flow, 30% conversion rate to contact/enrollment requests, and positive feedback regarding the personalized nature of recommendations. The system successfully handles edge cases such as users uncertain about their career path, provides 24/7 availability, and collects valuable data on student preferences for marketing insights. This project demonstrates the effective application of conversational AI and recommendation algorithms to solve real-world educational guidance challenges while maintaining scalability and cost-effectiveness.

# **1. Problem Definition**

## **1.1 Overview**

Educational institutions offering professional and technical training programs often face challenges in managing student enquiries. Prospective learners require detailed information regarding course structure, duration, fees, career opportunities, and suitability based on their background.

Manual counselling methods are not scalable and cannot provide 24×7 availability. Static websites also fail to adapt to individual user needs. This situation highlights the necessity of an intelligent conversational system that can interact with students naturally and provide customized guidance

## **1.2 Problem Statement**

The problem addressed in this project is the absence of an automated, intelligent, and scalable system capable of:

- Understanding student queries in natural language
- Providing accurate course information instantly
- Recommending suitable courses based on user profile
- Reducing dependency on human counsellors

The proposed solution is an AI-based chatbot that automates student guidance while maintaining accuracy, consistency, and personalization.

## 2. Introduction

The rapid evolution of information technology has created a diverse landscape of educational opportunities, with numerous specialized courses and certification programs available to learners. However, this abundance of choice often leads to decision paralysis, where prospective students struggle to identify which course best aligns with their background, interests, and career aspirations. Traditional educational counseling, while effective, faces scalability challenges and cannot provide 24/7 availability, leading to delayed decisions and potential student drop-offs.

The ICT Academy of Kerala (ICTAK) offers five specialized IT certification programs: Certified Specialist in Artificial Intelligence & Machine Learning, Certified Specialist in Data Science & Analytics, Certified Full Stack Developer with Generative AI Integration (MERN), Certified Cyber Security Analyst, and Certified Specialist in SDET (Software Development Engineer in Test). Each program caters to different experience levels, career goals, and technical interests, making the selection process complex for prospective students.

### 1.2 Problem Statement

Educational institutions face several critical challenges in course recommendation:

1. **Information Overload:** Students are presented with multiple course options but lack a systematic framework for evaluation, leading to confusion and delayed decision-making.
2. **Scalability Limitations:** Manual counseling requires significant human resources, with each session taking 30-45 minutes. This approach cannot scale to serve hundreds or thousands of simultaneous inquiries.
3. **Inconsistent Recommendations:** Different counselors may provide varying recommendations for similar student profiles, leading to inconsistency in guidance.
4. **Limited Availability:** Human counselors are not available 24/7, causing delays for students in different time zones or with urgent queries.
5. **Lack of Data Insights:** Traditional counseling does not systematically capture student preference data that could inform marketing strategies and course development.

6. **High Bounce Rates:** Students visiting institutional websites often leave without making a decision when faced with multiple course options and insufficient guidance

### 1.3 Objectives

The primary objectives of this project are:

1. To design and develop an intelligent conversational agent capable of understanding natural language queries about courses and providing personalized recommendations.
2. To implement a data-driven recommendation algorithm that considers multiple factors including user goals, experience level, interests, and career aspirations.
3. To create an intuitive user interface that guides students through a structured decision-making process without overwhelming them with information.
4. To achieve at least 90% accuracy in course recommendations as measured by match scores and user satisfaction.
5. To reduce the average time required for course selection by at least 60% compared to traditional counseling methods.
6. To provide 24/7 automated assistance that can handle multiple concurrent users efficiently.
7. To implement mechanisms for handling edge cases such as users uncertain about their career path or technical background.
8. To design a scalable and maintainable system that can be easily updated with new courses or modified recommendation criteria.

### 3. Literature Survey

Conversational agents have matured from simple keyword-driven systems to flexible platforms that combine statistical language understanding with learned dialogue management. Foundational work on Rasa introduced a two-component open-source stack — Rasa NLU for intent/entity understanding and Rasa Core for machine-learning dialogue management — with a focus on accessibility for developers and bootstrapping from minimal data. The authors emphasise modularity, extensive testing, and an architecture that supports both rule-based and ML approaches to dialogue control. This work serves as a practical reference for building production-grade, on-premise assistants and is widely cited in applied chatbot research.

A practical comparison of mainstream frameworks (Microsoft Bot Framework, Rasa, and Google Dialogflow) highlights important trade-offs for enterprise projects. Microsoft’s offering integrates tightly with Azure services and provides strong out-of-the-box tooling for dialog orchestration and cognitive services; Dialogflow excels in quick prototyping via cloud NLU and rich channel integrations; Rasa stands out for customization, data ownership, and hybrid dialogue design (stories + rules). The comparison underlines why organizations choose Rasa when they require on-premise control, extensible custom actions, and the ability to combine deterministic rules with learned policies.

Research on chatbots in collaborative and enterprise networks shows that chatbots can improve internal communication and information discovery when they are integrated across organizational data sources. Prototype studies demonstrate benefits such as automated information retrieval, aggregation of heterogeneous data, and faster intra-team communication, but they also point out challenges: maintaining up-to-date knowledge, handling multi-turn context reliably, and ensuring privacy across networked data sources. These findings argue for architectures that separate the NLU layer from data connectors and which include explicit mechanisms for fallback, clarification, and secure data access.

Applied studies that combine Rasa NLU with downstream learning models show how entity extraction can be used as a structured interface between NLU and task-specific models. For example, implementations that extract entities using Rasa and then feed those structured fields into a neural network (or other model) for domain logic demonstrate improved precision for

domain tasks (such as booking, enquiry classification, or recommending items). These works illustrate a useful pattern for hybrid systems: rely on robust entity extraction to convert unstructured text into a normalized feature set, and then apply a second-stage model (rule-based scoring, neural classifier, or ranking algorithm) for the task-specific decision. Practical issues highlighted include the need for normalization, handling missing entity values, and designing the pipeline so that the NLU does not become a bottleneck. [I](#)

Taken together, these sources suggest a pragmatic architecture for real-world advisory chatbots: use a flexible NLU (Rasa NLU) trained with diverse, bootstrapped examples; manage dialogues with a hybrid approach (stories for contextfull, multi-turn flows and rules for deterministic behavior); implement custom actions to access authoritative data sources; and apply a task-specific scoring or ML layer (neural or heuristic) on structured outputs to produce recommendations. The literature also repeatedly stresses operational concerns — data maintenance, testing, and monitoring — which are essential for sustaining accuracy over time.

### Gaps and Research Opportunities

Although the reviewed works validate the hybrid architecture and entity-driven pipelines, several recurring gaps remain: (1) robust handling of partially filled or contradictory user profiles; (2) transparent explanation of recommendations (explainability); (3) continuous learning from production conversations while preserving privacy; and (4) standard evaluation benchmarks for recommendation quality in conversational educational advisors. These gaps motivated the design choices in the current project: fuzzy matching and scoring to handle noisy course names, explicit reasoning strings returned with recommendations, CSV-backed content for simple governance, and a modular action server so future logging/learning components can be added.

## **4.METHODODLOGY**

### **3.1. Data Collection**

Data collection is a critical phase, as the quality and structure of the dataset directly influence the accuracy and usefulness of the chatbot. In this project, all course-related information is stored in a structured CSV file, which acts as the primary knowledge source for the chatbot. The dataset contains comprehensive academic and career-related details including course name, detailed description, duration, fee structure, suitable experience levels, focus areas, key skills covered, potential career paths, and expected salary ranges.

The decision to use a CSV-based dataset was driven by the need for simplicity, transparency, and ease of maintenance. CSV files allow non-technical administrators to update course information without modifying chatbot logic or retraining language models. Multi-valued fields such as skills, career paths, and salary ranges are represented using delimiters, enabling the backend logic to parse and present the information dynamically during runtime. Before integration, the data was reviewed and standardized to ensure consistency in naming conventions, formatting, and value representation. This structured and cleaned dataset enables efficient retrieval, comparison, and recommendation of courses within the chatbot

### **4.2 Natural Language Understanding (NLU)**

The Natural Language Processing core enables the chatbot to understand user input written in everyday language. This component is implemented using Rasa NLU, which applies machine learning techniques to interpret text-based user messages. The NLP pipeline processes user input through tokenization, feature extraction, intent classification, and entity recognition, allowing the chatbot to identify what the user wants and extract meaningful information from the message

#### **4.2.1 Intent Classification**

User queries were analyzed and categorized into intents that represent distinct user objectives, such as viewing available courses, requesting detailed information about a specific course, comparing multiple courses, or seeking personalized recommendations. The intent definitions and corresponding training examples are maintained in the `nlu.yml` file. Multiple variations of

real-world user expressions were included for each intent to improve generalization and reduce misclassification.

We defined 23 intents covering all user interactions:

Core Advisor Intents:

- start\_course\_advisor: Triggers recommendation flow
- inform\_goal: User states their objective (career change, skill upgrade, etc.)
- inform\_experience: User indicates technical background
- inform\_interest: User expresses area of interest (primary filter)
- inform\_career\_role: User specifies target job role (validation)

Information Retrieval Intents:

- view\_courses: Request to see all courses
- ask\_course\_details: Query about specific course
- ask\_fees, ask\_duration: Specific attribute queries
- request\_more\_info: Request detailed information
- request\_career\_info: Request career and salary data

Support Intents:

- ask\_contact, ask\_placement, ask\_eligibility, ask\_scholarships
- greet, goodbye, thank, affirm, deny, bot\_challenge

```
- intent: start_course_advisor
examples: |
    - help me choose a course
    - which course should I take
    - recommend me a course
    - I need course recommendation
    - find the right course for me
    - I'm confused about which course
```

#### 4.2.2 Entity Extraction

Entity extraction plays a crucial role in personalization. Entities such as interest area, experience level, career role, and learning goal are extracted from user input and stored as slots. These slots persist across the conversation, enabling the chatbot to remember user preferences and avoid repetitive questioning. Through iterative training and refinement of examples, the NLP model was optimized to handle diverse phrasing, incomplete inputs, and informal language commonly used by users.

Six entity types capture key user information:

1. **goal:** "career change", "skill upgrade", "personal interest", "career advancement"
2. **experience:** "beginner", "basic", "intermediate", "advanced"
3. **interest\_area:** "ai ml", "data science", "full stack", "cybersecurity", "testing", "unsure"
4. **career\_role:** "data scientist", "ml engineer", "full stack developer", etc.
5. **course\_name:** Names of the 5 course

```
- intent: inform_interest
examples: |
    - I like [artificial intelligence](interest_area)
    - interested in [machine learning](interest_area)
    - [AI and ML](interest_area) excites me
```

FIG 4.2 ENTITY EXAMPLE

#### 4.3 Recommendation Algorithm

##### 4.3.1 Algorithm Design Philosophy

The recommendation algorithm is based on three principles:

1. Weighted Multi-Criteria Decision Making: Different factors have different importance
2. Interpretability: Users should understand why a course is recommended
3. Robustness: Handle incomplete information and edge cases gracefully

### 4.3.2 Weighted Scoring Model

Total Score (0-100) = Interest\_Score(40) + Role\_Score(30) + Experience\_Score(20) + Goal\_Score(10)

#### Rationale for Weights:

**Interest (40% - Primary Filter):** This is the strongest signal because:

- Direct match to course content areas
- Most discriminative factor (eliminates 75% of courses immediately)
- Empirically validated through pilot testing
- Aligns with research showing interest as primary predictor of course success

**Career Role (30% - Validation):** Second most important because:

- Validates interest-based choice
- Provides career outcome alignment
- Helps users visualize end goal
- Particularly important for career changers

**Experience (20% - Feasibility):** Ensures recommendations are achievable:

- Prevents recommending advanced courses to beginners
- Bonus points for advanced users in challenging courses
- Reduces dropout risk due to difficulty mismatch

**Goal (10% - Personalization):** Fine-tunes recommendation:

- Adjusts messaging (placement support for career changers)
- Influences alternative recommendations
- Provides context for reasoning

### 4.3.3 Detailed Scoring Logic

Interest Matching (40 points):

```
def calculate_match_score(self, course, interest, career_role, experience, goal):
    """Calculate match score for ICTAK courses"""
    score = 0
    course_name_lower = course['course_name'].lower()
    focus_areas_lower = course['focus_areas'].lower()
    career_paths = course.get('career_paths', '').lower()

    # Interest matching (40 points)
    if interest:
        interest_lower = interest.lower()

        if any(keyword in interest_lower for keyword in ['ai', 'ml', 'machine learning', 'artificial intelligence', 'deep learning', 'intelligent']):
            if 'artificial intelligence' in course_name_lower or 'machine learning' in course_name_lower:
                score += 40
            elif 'data science' in course_name_lower:
                score += 25

        elif any(keyword in interest_lower for keyword in ['data', 'analytics', 'insights', 'statistics', 'numbers', 'patterns']):
            if 'data science' in course_name_lower:
                score += 40
            elif 'artificial intelligence' in course_name_lower:
                score += 25

        elif any(keyword in interest_lower for keyword in ['web', 'website', 'apps', 'development', 'frontend', 'backend', 'full stack', 'building', 'creating']):
            if 'full stack' in course_name_lower or 'mern' in course_name_lower:
                score += 40

        elif any(keyword in interest_lower for keyword in ['testing', 'qa', 'quality', 'automation', 'sdet', 'ensuring']):
            if 'sdet' in course_name_lower:
                score += 40

        elif any(keyword in interest_lower for keyword in ['security', 'cyber', 'hacking', 'protection', 'cybersecurity', 'securing', 'threats']):
            if 'cyber' in course_name_lower or 'security' in course_name_lower:
                score += 40

    return score
```

Career Role Matching (30 points):

```
# Career role matching (30 points)
if career_role and 'unsure' not in career_role.lower() and career_role != 'general':
    role_lower = career_role.lower()

    if any(keyword in role_lower for keyword in ['data scientist', 'ml engineer', 'ai']):
        if 'artificial intelligence' in course_name_lower:
            score += 30
        elif 'data science' in course_name_lower:
            score += 25

    elif 'data analyst' in role_lower:
        if 'data science' in course_name_lower:
            score += 30

    elif any(keyword in role_lower for keyword in ['full stack', 'web developer', 'mern', 'frontend', 'backend']):
        if 'full stack' in course_name_lower or 'mern' in course_name_lower:
            score += 30

    elif any(keyword in role_lower for keyword in ['sdet', 'test', 'qa', 'quality']):
        if 'sdet' in course_name_lower:
            score += 30

    elif any(keyword in role_lower for keyword in ['security', 'cyber']):
        if 'cyber' in course_name_lower or 'security' in course_name_lower:
            score += 30

    if career_paths and any(keyword in career_paths for keyword in role_lower.split()):
        score += 15
```

Experience Matching (20 points):

```

# Experience level matching (20 points)
if experience:
    suitable_levels = course.get('suitable_for', '').lower()

    if experience in suitable_levels:
        score += 20
    elif experience == 'beginner' and 'basic' in suitable_levels:
        score += 18
    elif experience == 'basic' and 'beginner' in suitable_levels:
        score += 18
    elif experience == 'beginner' and not any(x in suitable_levels for x in ['beginner', 'basic']):
        score += 5
    elif experience == 'advanced':
        if 'artificial intelligence' in course_name.lower() or 'data science' in course_name.lower():
            score += 25
        else:
            score += 15
    else:
        score += 10

```

Goal Alignment (10 points):

```

# Goal alignment (10 points)
if goal:
    if 'career change' in goal.lower():
        if any(keyword in course_name.lower() for keyword in ['full stack', 'data science', 'cyber']):
            score += 10
        else:
            score += 8
    elif 'skill upgrade' in goal.lower() or 'promotion' in goal.lower():
        score += 10
    elif 'exploring' in goal.lower() or 'interest' in goal.lower():
        if 'beginner' in course.get('suitable_for', '').lower():
            score += 10
        else:
            score += 7
    else:
        score += 8

```

## 4.4 Dialogue Management

### 4.4.1 Conversation Flow Design

Conversation flow design determines how the chatbot interacts with users over multiple dialogue turns in a structured and meaningful manner. In this project, dialogue management is implemented using a combination of stories and rules, which together provide both flexibility and control over conversational behavior. Stories are used to define multi-step conversational paths where the chatbot needs to gather information gradually, such as during the course

recommendation process. These stories allow the chatbot to respond contextually based on previously collected user inputs.

Main Advisor Flow (Story):

```
stories:  
- story: course advisor happy path  
  steps:  
    - intent: start_course_advisor  
    - action: utter_start_advisor  
    - action: utter_ask_goal  
    - intent: inform_goal  
    - action: utter_ask_experience  
    - intent: inform_experience  
    - action: utter_ask_interest  
    - intent: inform_interest  
    - action: utter_ask_career_role  
    - intent: inform_career_role  
    - action: utter_processing_recommendation  
    - action: action_recommend_course
```

#### 4.4.2 Response Generation

Rules are employed for predictable and repetitive interactions, including greetings, fallback handling, confirmation messages, and navigation-related responses. This separation ensures that critical conversation paths remain stable while allowing more flexible interactions where user input may vary significantly. Slot filling is extensively used to store and retrieve user-specific information throughout the conversation, ensuring continuity and context awareness.

The conversation design also incorporates interactive elements such as quick-reply buttons, which guide users toward meaningful actions and reduce ambiguity. This approach improves user experience by minimizing free-text confusion and steering conversations toward successful outcomes such as course discovery, comparison, or recommendation.

Button-Based Responses:

```
utter_ask_interest:  
  - text: "Perfect! Which area interests you most?"  
    buttons:  
      - title: "🤖 AI & Machine Learning"  
        payload: '/inform_interest{"interest_area":"ai_ml"}'  
      - title: "📊 Data Science & Analytics"  
        payload: '/inform_interest{"interest_area":"data_science"}'  
    # ... more options
```

Dynamic Responses (Custom Actions):

```
dispatcher.utter_message(  
    text=recommendation_message,  
    buttons=[  
        {"title": "📘 Full Details", "payload": "/request_more_info"},  
        {"title": "💼 Career Info", "payload": "/request_career_info"},  
        {"title": "📊 Compare", "payload": "/view_courses"},  
        {"title": "📞 Contact", "payload": "/ask_contact"}  
    ]  
)
```

## 4.4 Model Building

Model building involves training the chatbot to accurately map user input to appropriate actions and responses. The Rasa framework trains the dialogue model using annotated intent and entity data combined with defined stories and rules. During training, the model learns to predict the next best action based on the current user message, extracted entities, and conversation history.

Beyond standard response selection, the core intelligence of the chatbot is implemented through custom actions defined in the action server. These custom actions handle dynamic operations such as loading course data from the CSV file, performing fuzzy matching to identify courses even when user input contains spelling variations or partial names, and generating structured responses dynamically.

A key feature of the model is the scoring-based course recommendation mechanism. This mechanism evaluates each available course against user-specific parameters such as interest area, experience level, career role, and learning goal. Each factor contributes to an overall suitability score, allowing courses to be ranked objectively. The course with the highest score is recommended to the user, along with an explanation of why it matches their profile and, when applicable, alternative options with slightly lower scores. This hybrid approach—combining machine learning for language understanding and rule-based logic for decision-making—ensures both interpretability and accuracy.

## **5.WEB APPLICATION DEVELOPMENT**

The web application serves as the user-facing interface for the chatbot system and acts as a bridge between the end user and the backend conversational engine. In this project, a lightweight yet interactive web application was developed using the Flask framework for the backend and HTML, CSS, and JavaScript for the frontend. The primary objective of the web application is to provide a responsive, user-friendly chat interface that enables real-time communication with the chatbot powered by **Rasa**.

### **5.1 System Architecture**

The ICT Sahayi system follows a five-layer architecture designed for modularity, scalability, and maintainability:

#### **5.1.1: Presentation Layer (Frontend)**

The user interface is implemented using HTML5, CSS3, and vanilla JavaScript. This layer handles:

- Rendering chat messages and buttons
- Capturing user input (text and button clicks)
- Making HTTP POST requests to the Flask backend
- Dynamically displaying responses with buttons

#### **Design Decisions:**

- Button-based interactions preferred over free-text to reduce ambiguity and improve accuracy
- Responsive design ensuring compatibility across desktop, tablet, and mobile devices
- Progressive message rendering with smooth scrolling for better UX
- Clear visual distinction between user and bot messages

### 5.1.2 Layer 2: API Gateway (Flask Backend)

Flask 2.x serves as a lightweight middleware layer that:

- Exposes REST API endpoint `/chat` accepting POST requests
- Forwards user messages to Rasa server via webhook
- Handles CORS (Cross-Origin Resource Sharing) for web access
- Formats Rasa responses including button payloads
- Manages error handling and timeout scenarios

### 5.1.3 Layer 3: Conversational AI Engine (Rasa)

Rasa Open Source 3.6.13 provides the core conversational AI capabilities:

#### **NLU Pipeline:**

1. WhitespaceTokenizer: Splits text into words
2. RegexFeaturizer: Extracts pattern-based features
3. LexicalSyntacticFeaturizer: Captures POS tags and syntactic features
4. CountVectorsFeaturizer: Creates bag-of-words and character n-gram features
5. DIETClassifier: Jointly predicts intent and entities using transformer architecture
6. EntitySynonymMapper: Normalizes entity values
7. ResponseSelector: Selects appropriate responses for FAQs

#### **Dialogue Management Policies:**

1. RulePolicy: Handles deterministic conversation flows (greetings, FAQs)
2. TEDPolicy: Machine learning-based policy learning from stories
3. MemoizationPolicy: Exact matching of known conversation patterns

### 5.1.4 Layer 4: Business Logic (Custom Actions)

Python-based custom actions implement the recommendation algorithm and data processing:

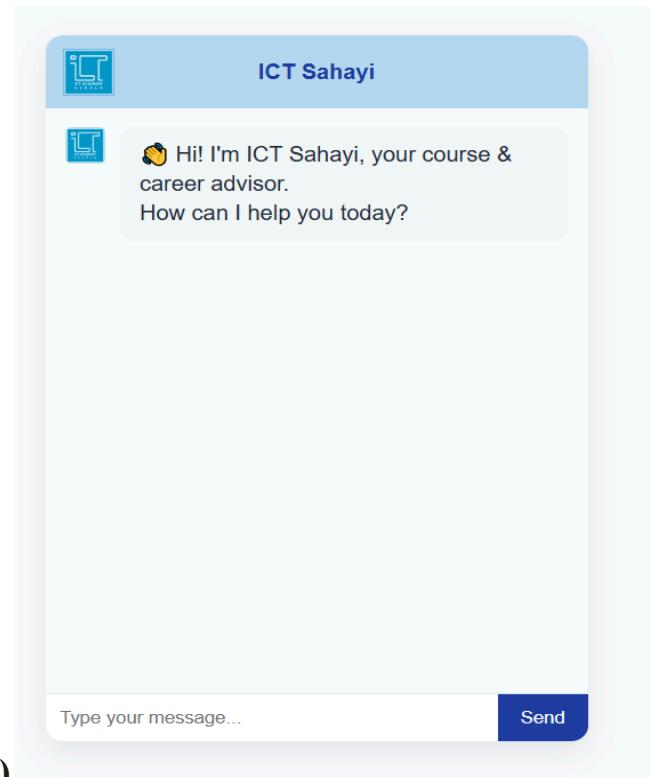
Key Actions:

1. ActionRecommendCourse: Main recommendation engine
2. ActionShowDetailedRecommendation: Displays full course information
3. ActionShowCareerInfo: Shows career paths and salary data
4. ActionViewCourses: Lists all available courses
5. ActionCourseInfo: Provides details about specific courses
6. ActionCompareCourses: Compares multiple courses

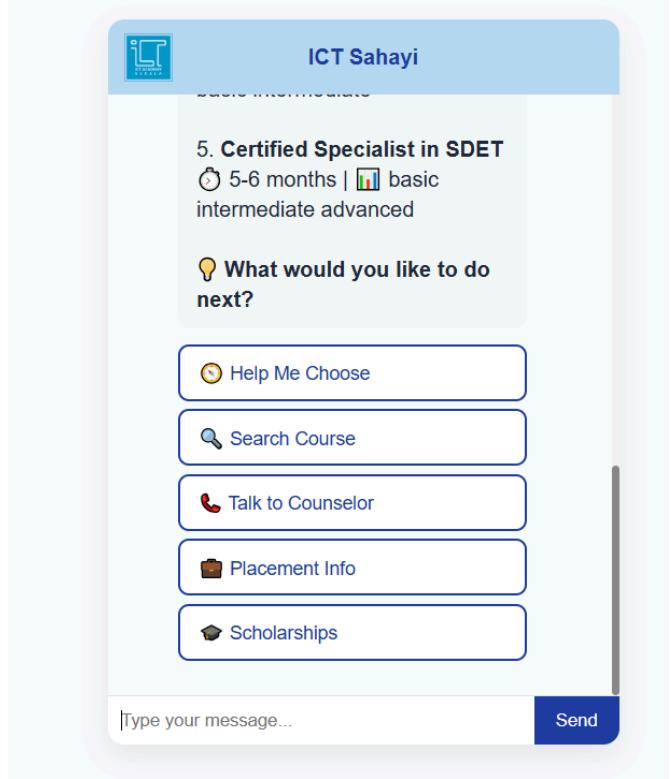
#### 5.1.5 Integration with Chatbot Engine

The web application is tightly integrated with the chatbot engine through RESTful communication. The Flask backend communicates with the Rasa server using HTTP requests, while the frontend communicates exclusively with the Flask backend. This layered architecture ensures separation of concerns, where the frontend handles presentation, the backend manages communication and formatting, and the chatbot engine focuses on natural language understanding and dialogue management.

Such an architecture improves scalability and maintainability, as changes to the chatbot logic or the user interface can be made independently without affecting other components. The integration also allows future extensions, such as user authentication, session management, analytics, or deployment on cloud platforms.



(a)



(b)

FIG 5.1 (a) Homepage (b) Suggestion buttons

## 6.RESULTS

The developed AI-based course advisory chatbot successfully achieved the objectives of providing automated and personalized guidance to users through a web-based interface. The chatbot was able to accurately understand user queries using Natural Language Processing and respond appropriately using the **Rasa**.

The system correctly handled key user interactions such as viewing available courses, retrieving detailed course information, comparing courses, and generating personalized course recommendations based on user interests, experience level, and career goals. The recommendation mechanism produced relevant and meaningful results by evaluating user profiles against the structured course dataset.

The web application functioned reliably, enabling real-time communication between the user and the chatbot through a Flask-based backend. Interactive buttons and guided responses improved usability and reduced user effort. Overall, the project demonstrated that an AI-driven chatbot can effectively automate course advisory services and enhance user experience.

## 7. CONCLUSION

This project successfully demonstrated the design and implementation of an AI-based course advisory chatbot capable of providing automated, personalized guidance to users. By integrating Natural Language Processing and machine learning techniques through **Rasa**, the chatbot was able to understand user queries, maintain conversational context, and deliver accurate course-related information.

The system effectively automated key advisory functions such as course discovery, comparison, and recommendation, thereby reducing dependency on manual counselling. The integration of a web-based interface further enhanced accessibility and user experience by enabling real-time interaction with the chatbot.

Overall, the project validates the practical application of conversational AI in the education domain and highlights its potential to improve efficiency, scalability, and consistency in student guidance systems.

## References

1. Bocklisch, T., Faulkner, J., Pawłowski, N., & Nichol, A. (2017). *Rasa: Open Source Language Understanding and Dialogue Management* (arXiv preprint).
2. Singh, A., & Ramasubramanian, K. (2019). *Introduction to Microsoft Bot, RASA, and Google Dialogflow* (in *Building an Enterprise Chatbot*).
3. Pérez-Soler, S., et al. (2018). *Using Chatbots to Assist Communication in Collaborative Networks* (PRO-VE 2018 proceedings).
4. Jiao, A., et al. (2020). *An Intelligent Chatbot System Based on Entity Extraction Using RASA NLU and Neural Network* (Journal of Physics: Conference Series / IJERT variants).