**C. K. Pithawala College of Engineering and Technology, Surat**

**Course-Integrated Practical & Analytical Task (CIPAT)**

# Prolog-based Student Course Advisor

**A PROJECT REPORT**

***Submitted by***

**Anishree K Patwa**
**(220090107005)**

*Prepared as a part of the requirements for the subject of*

**Artificial Intelligence(3170716)**

**BACHELOR OF ENGINEERING**

***in***

**Computer Engineering IV Year , Semester- VII**

**C. K. Pithawala College of Engineering and Technology, Surat**

**Gujarat Technological University, Ahmadabad**

**September, 2025**

**Prof. Gira Barot**
Asst. Professor

**Dr. Vishruti V. Desai**
Head of the Departmen

# Table of Content

# List of Figures

# Abstract

The project presents a rule-based advisor system developed in Prolog that recommends elective courses to students based on their academic percentage, subject interests, and career goals. The system uses predicate logic and search techniques to identify valid courses from the knowledge base. The project also investigates the application of Artificial Intelligence (AI) in modern education, focusing on personalized learning paths, dropout prediction, and student performance analytics.

This approach highlights how classical AI methods, such as rule-based reasoning, can still provide effective solutions for decision support. At the same time, it contrasts with modern data-driven AI techniques, showing the evolution of educational technologies. The system demonstrates the importance of logic programming in building transparent, explainable models while paving the way for integration with advanced AI for more dynamic and scalable applications.

# 1. Introduction

Course selection plays a crucial role in shaping a student's academic and professional journey. Traditionally, students rely on faculty advisors, but these decisions are often subjective and may not fully align with the student's performance, interests and long-term goals.

Artificial Intelligence offers a systematic and data-driven way to handle such problems by reducing subjectivity and improving decision-making. Prolog, with its strong foundation in predicate logic, rule-based reasoning and pattern matching, is particularly well-suited for building expert systems such as course advisors.

A rule-based advisor system can evaluate multiple factors—such as academic percentage, preferred subjects and career aspirations—and then apply logical inference to recommend the most suitable electives. Such a system not only saves time for both students and faculty but also ensures that recommendations are consistent, transparent and personalized.

Furthermore, the integration of AI into education is not limited to course recommendation. Universities across the world are adopting intelligent systems for tasks such as personalized learning path generation, predicting student dropout risks and analyzing performance trends. These applications highlight the growing role of AI in transforming education from a one-size-fits-all model to a more adaptive, learner-centric approach.

## 2. Problem Statement

Students often face difficulty in identifying electives that align with their overall profile. The key challenges include:

- **Interests** (e.g., Artificial Intelligence, Cloud Computing, Information Security).
- **Academic performance** (measured through percentage/grades).
- **Future career goals** (e.g., researcher, cloud engineer, cybersecurity analyst).

To address this, the objective of the project is to develop a **Prolog-based Student Course Advisor**. The system leverages **facts, rules and constraints** to evaluate a student's profile and recommend the most suitable elective courses. By combining logical inference with structured decision-making, the advisor ensures that the selected courses are not only academically appropriate but also aligned with the student's long-term aspirations.

# 3. Methodology

## • Search Techniques

Depth-First Search (DFS) is a systematic method of exploring a search space by going as deep as possible along one branch before moving to another. It works like following a single path until no further progress can be made, then "backtracking" to the last decision point and trying a different path.

DFS process:

- Start at the root (initial state).
- Explore one branch fully before moving to the next.
- If a dead-end is reached (no solution), return to the last unexplored branch (backtracking).
- Continue until a solution (or all solutions) are found.

Backtracking is the key feature—it ensures that when one possible path fails, the algorithm returns to previous steps and explores alternative paths, thereby covering the entire search space.

**How Prolog Uses DFS with Backtracking :**

Prolog internally applies DFS with backtracking when answering queries.

Example Query: recommend_course(alice, Course).

- Prolog first searches all **interested_in/2** facts for Alice.
- It then tries to match these with **course/3** facts.
- Next, it checks the percentage constraint to see if Alice is eligible.
- If a course does not satisfy all conditions, Prolog backtracks to try another possible course.

This ensures that Prolog is able to return not only the first valid recommendation but all possible courses that satisfy Alice's profile.

- ## System Implementation

### 1. Knowledge Base

- The system stores **facts** about students, their percentages, interests, and future goals.
- Courses are also defined with minimum percentage requirements.
  Lets say for example ,
  student(alice).
  percentage(alice, 85).
  interested_in(alice, ai).
  goal(alice, researcher).
  course(ai_basics, ai, 70).
  course_goal_match(ai_basics, researcher).
- This structure makes the knowledge explicit and machine-readable.

### 2. Recommendation Rules

- Two logical rules are used to suggest courses:
  - **Interest + Percentage Match**
- A course is suggested if the student is interested and has required marks.
  - **Goal-Aware Match**
- A course is also suggested if it matches both interest and career goal.
  Example Rule:

```
recommend_course(Student, Course) :-
    student(Student),
    interested_in(Student, Interest),
    percentage(Student, Perc),
    course(Course, Interest, MinPerc),
    Perc >= MinPerc.
```

### 3. Utility Predicates

- Utilities make the system user-friendly.
- get_interests/2 → Collects all interests of a student.
- get_goal/2 → Fetches student's career goal.
- get_percentage/2 → Retrieves marks.
- suggest_courses/2 → Collects all valid course recommendations.
  Example:
  ?- suggest_courses(alice, Courses).

Courses = [ai_basics, machine_learning, data_science].

4. **Student Details & Printing**

- print_student_details/1 → Shows full details of one student.
- print_all_students/0 → Displays summary of all students.
- print_table/0 → Prints results in a **single-line table format** (easy to read).
- Example Output (table):

5. **CSV Export**

   - The system can also export results to a .csv file for external use.
   - This shows integration of Prolog with real-world data reporting.

     Example:

     ?- print_csv('students_summary.csv').

   - This above code generatesa CSV file with all student details and recommendations.

6. **Search Techniques in Action**

- The system internally uses **backtracking and search** to find matching rules.
- Example: Prolog tries all possible interests and courses, checks constraints, and outputs valid recommendations.
- Hence, demonstrates **search + predicate logic + knowledge representation**

| Student | Interest(s) | % | Goal | Suggested Courses |
|---|---|---|---|---|
| Alice | AI, ML | 85 | Researcher | ai_basics, machine_learning, data_science |
| Bob | Cloud | 60 | Cloud Engineer | [] (not eligible) |
| Charlie | Security | 75 | Security Expert | cyber_security |

```
% ------------------ Knowledge Base ------------------

% Students
student(alice).
student(bob).
student(charlie).

% Student percentage
percentage(alice, 85).
percentage(bob, 60).
percentage(charlie, 75).

% Student interests
interested_in(alice, ai).
interested_in(alice, ml).
interested_in(bob, cloud).
interested_in(charlie, security).

% Student future goals
goal(alice, researcher).
goal(bob, cloud_engineer).
goal(charlie, security_expert).

% Available courses
course(ai_basics, ai, 70).
course(machine_learning, ml, 75).
course(cloud_computing, cloud, 65).
course(cyber_security, security, 70).
course(data_science, ai, 80).

% Map courses to future goals
course_goal_match(ai_basics, researcher).
course_goal_match(machine_learning, researcher).
course_goal_match(cloud_computing, cloud_engineer).
course_goal_match(cyber_security, security_expert).
course_goal_match(data_science, researcher).



% ------------------ Recommendation Rules ------------------

% Recommend if interest & percentage match
recommend_course(Student, Course) :-
    student(Student),
    interested_in(Student, Interest),
    percentage(Student, Perc),
    course(Course, Interest, MinPerc),
    Perc >= MinPerc.

% Also allow goal-aware match (this may cause duplicate matches without list_to_set)
recommend_course(Student, Course) :-
    student(Student),
    interested_in(Student, Interest),
    goal(Student, Goal),
    percentage(Student, Perc),
    course(Course, Interest, MinPerc),
    Perc >= MinPerc,
    course_goal_match(Course, Goal).
```

```
% ------------------ Utilities ------------------

% Collect recommendations and remove duplicates
suggest_courses(Student, Courses) :-
    findall(Course, recommend_course(Student, Course), List),
    list_to_set(List, Courses).

% collect all interests of a student as a set
get_interests(Student, Interests) :-
    findall(I, interested_in(Student, I), List),
    list_to_set(List, Interests).

% safe getter for goal (returns 'none' if no goal)
get_goal(Student, Goal) :-
    ( goal(Student, G) -> Goal = G ; Goal = none ).

% safe getter for percentage (returns 0 if not present)
get_percentage(Student, P) :-
    ( percentage(Student, X) -> P = X ; P = 0 ).

% print list of items as comma separated (no brackets)
print_list([]) :- write('None').
print_list([H]) :- write(H).
print_list([H|T]) :-
    write(H), write(', '), print_list(T).



% ------------------ Print / Table ------------------

% print single student full details
print_student_details(Student) :-
    format('~nStudent: ~w~n', [Student]),
    get_interests(Student, Interests),
    write(' Interests: '), print_list(Interests), nl,
    get_percentage(Student, Perc),
    format(' Marks: ~w~n', [Perc]),
    get_goal(Student, Goal),
    format(' Goal: ~w~n', [Goal]),
    suggest_courses(Student, Courses),
    write(' Suggested Courses: '),
    ( Courses = [] -> write('None') ; print_list(Courses) ),
    nl.

% print summary for all students (verbose block format)
print_all_students :-
    total_students(N),
    format('~nTotal students: ~w~n', [N]),
    format('------------------------------------------------~n'),
    student(S),
    print_student_details(S),
    fail.
print_all_students :-
    format('------------------------------------------------~n'),
    true.

% count distinct students
total_students(N) :-
    findall(S, student(S), L),
    list_to_set(L, Set),
    length(Set, N).



% pretty table (single-line per student)
print_table :-
    format('~n-------------------------------------------------------------~n'),
    format('| Student    | Interests          | Marks | Goal              | Suggested Courses               |~n'),
    format('-------------------------------------------------------------~n'),
    student(S),
    get_interests(S, Interests),
    get_percentage(S, Perc),
    get_goal(S, Goal),
    suggest_courses(S, Courses),
    format('| ~w~t~11+ | ', [S]),
    ( Interests = [] -> write('None') ; format('~w~t~20+', [Interests]) ),
    format(' | ~w    | ~w~t~20+ | ', [Perc, Goal]),
    ( Courses = [] -> write('None') ; print_list(Courses) ),
    nl,
    fail.
print_table :-
    format('-------------------------------------------------------------~n').

% ---------------- CSV Export ----------------
```

```
% ------------------ CSV Export ------------------

% write a header and all student rows to students_summary.csv
print_csv(FileName) :-
    open(FileName, write, Stream),
    write(Stream, 'Student,Interests,Marks,Goal,SuggestedCourses'), nl(Stream),
    student(S),
    get_interests(S, Interests),
    get_percentage(S, Perc),
    get_goal(S, Goal),
    suggest_courses(S, Courses),
    % convert lists to comma-free strings (join with ';' to avoid CSV conflict)
    list_to_string_with_sep(Interests, ';', IntStr),
    list_to_string_with_sep(Courses, ';', CourseStr),
    format(Stream, '~w,~w,~w,~w,~w~n', [S, IntStr, Perc, Goal, CourseStr]),
    fail.
print_csv(FileName) :-
    % this clause is reached after fail backtracks exhausted
    open(FileName, append, Stream), close(Stream), !.  % ensure file closed

% helper: join list elements into string separated by Sep
list_to_string_with_sep([], _, '').
list_to_string_with_sep([H], _, Str) :-
    atom_string(H, Str).
list_to_string_with_sep([H|T], Sep, Str) :-
    atom_string(H, Hs),
    list_to_string_with_sep(T, Sep, Ts),
    ( Ts = '' -> string_concat(Hs, Ts, Str) ; string_concat(Hs, Sep, Temp), string_concat(Temp, Ts, Str) ).

% ------------------ End of File ------------------
```

Figure.1:Program

- **Logic & Knowledge Representation**

  Here's a logical and Knowledge Representation of Prolog based Course Advisor:

  **Implementation Rules**

  **Rule 1: Interest + Percentage Match**

  - recommend_course(S, C) :-
  - student(S), interested_in(S, I),
  - percentage(S, P), course(C, I, M), P >= M.

  - **Example**: Alice (85%) interested in AI → eligible for ai_basics (70%) and data_science (80%).
  - **Why it matters**: Ensures students take courses they are both interested in and academically prepared for.

  **Rule 2: Goal-Oriented Recommendation**

  - recommend_course(S, C) :-

  - student(S), interested_in(S, I), goal(S, G),
  - percentage(S, P), course(C, I, M), P >= M,
  - course_goal_match(C, G).

  - **Example**: Charlie wants to be a security expert → recommended cyber_security (70%) since it aligns with his goal and he has 75%.

- **Why it matters**: Aligns academic choices with **future career paths**, not just marks.

Together, these rules make the advisor **personalized, logical and career-focused**.

## • AI Applications in Education

Real-world universities and online learning platforms are increasingly applying **Artificial Intelligence (AI) and Machine Learning (ML)** to enhance education:

### ➢ Personalized Learning Paths

- AI-driven adaptive systems recommend courses and study plans tailored to each learner's needs.
- By analyzing past performance, learning speed, and preferences, these systems suggest the most relevant next steps.
- *Example:* Coursera and Khan Academy use ML algorithms to recommend the next lesson or practice exercise for individual learners.

### ➢ Dropout Prediction

- ML models analyze student data such as attendance, grades, and online engagement patterns.
- These models can identify students at risk of dropping out and alert faculty to take early interventions.
- *Example:* Many universities use predictive analytics to reduce dropout rates by offering timely counseling and support.

### ➢ Performance Analytics

- AI-powered dashboards provide teachers with real-time insights into student progress.
- Instructors can identify weak areas, personalize feedback, and improve course effectiveness.
- *Example:* Georgia Tech and Stanford have experimented with AI tutors and analytics dashboards to monitor and support student learning.

# 4. Query Demonstration

The system was tested using different queries in SWI-Prolog. Queries such as retrieving suggested courses for a specific student, checking recommendations based on interests and goals, and printing all student details were successfully executed. The outputs confirmed that the rule-based system provides accurate and personalized elective course suggestions.

```
?- cd('C:/Users/Admin/OneDrive/Documents/SEMESTER 7/ARTIFICIAL INTELLIGENCE/prolog').
true.

?- pwd.
% c:/users/admin/onedrive/documents/semester 7/artificial intelligence/prolog/
true.

?- [advisor].
true.

?- print_all_students.

Total students: 3
------------------------------------------------

Student: alice
 Interests: ai, ml
 Marks: 85
 Goal: researcher
 Suggested Courses: ai_basics, data_science, machine_learning
machine_learning, None
ml, None
 Marks: 85
 Goal: researcher
 Suggested Courses: ai_basics, data_science, machine_learning
machine_learning, None

Student: bob
 Interests: cloud
 Marks: 60
 Goal: cloud_engineer
 Suggested Courses: None
cloud, None
 Marks: 60
 Goal: cloud_engineer
 Suggested Courses: None

Student: charlie
 Interests: security
 Marks: 75
 Goal: security_expert
 Suggested Courses: cyber_security
cyber_security, None
security, None
 Marks: 75
 Goal: security_expert
 Suggested Courses: cyber_security
cyber_security, None
------------------------------------------------
true.

?- print_table.

------------------------------------------------------------
| Student    | Interests          | Marks | Goal             | Suggested Courses                |
------------------------------------------------------------
| alice      | [ai,ml] | 85    | researcher | ai_basics, data_science, machine_learning
machine_learning, None
| bob        | [cloud] | 60    | cloud_engineer | None
| charlie    | [security] | 75    | security_expert | cyber_security
cyber_security, None
------------------------------------------------------------
true.

?- print_csv('students_summary.csv').
true.
```

Figure.2:Queries

## 5. Comparision with Modern System

- The **Prolog-based system** recommends electives using **explicit logical rules** (percentage, interests, career goals).
- **Modern educational AI systems** (e.g., Coursera, edX, university dashboards) use **machine learning** for:
- Personalized learning paths
- Dropout prediction
- Student performance analytics
- **Difference:** Prolog ensures **clear reasoning** and **explainability**, while modern AI ensures **adaptivity and scalability** with big data.
- **Together:** Rule-based systems are ideal for **structured, small-scale advising**, while modern AI systems support **dynamic, large-scale education platforms**.

## 6. Conclusion

The Prolog-based Student Course Advisor successfully recommends elective courses by applying logical rules, constraints, and backtracking search. The project highlights the effectiveness of **predicate logic** and **search techniques** in solving real-world decision-making problems within education.

This work also demonstrates how **AI applications** can bring structure, transparency and personalization to academic advising.

**Future Scope**

The system can be further enhanced by integrating **machine learning models** to provide data-driven recommendations, adapting to dynamic student data and offering real-time advising support similar to intelligent tutoring systems used in modern universities.

## 7. References


- SWI-Prolog Official Documentation – *https://www.swi-prolog.org/*
- GeeksforGeeks – *Prolog Basics and Examples – https://www.geeksforgeeks.org/prolog-introduction/*
- Coursera & Khan Academy – *Applications of AI/ML in Education*
- Georgia Tech AI Tutoring Research – *AI Applications in Higher Education*