Submitted to the Department of

Software Technology De La Salle – Manila

First Term, AY 2024-2025

Section S17

As one of the Required Outputs of the Course

Introduction to Intelligent Systems

MCO2: ChatBot

Submitted by:

Copon, Matthew Joshua

Fabiaña, Joshua Jose

Kelsey, Gabrielle Madison

Namuag, Leigh Avegail

Perdon, Linus Carl

Submitted to:

Norshuhani Zamin, Ph.D.

November 28, 2024

I. INTRODUCTION

Chatbots have increasingly gained popularity over the past few years. The widespread implementation of general-purpose AI like ChatGPT, Google Bard, Amazon Alexa, and Apple Siri made the public aware of its capabilities. Their scalability and faster response time are ideal for businesses that handle volumes of queries.

Beyond its business applications, Chatbots have a broad range of applications ranging from customer support to entertainment. Most of the popular chatbots nowadays are trained using machine learning. They use massive amounts of data from across the web and develop patterns to generate responses to user queries. Unlike this data-driven, bottom-up approach, rule-based chatbots operate using a set of rules in which data is to be processed. While machine learning is well-suited for general-purpose AI chatbots, simpler applications, like FAQ and customer support bots, often only require basic dialogues that can be linked to preset rules.

Determining relationships between members of a family tree operates with a specific set of rules. For example, each member is either a parent or a child of another. The common ancestor determines the relationship of any two members. A rule-based artificial intelligence operates best in this type of environment. In this project, an AI chatbot prompts the user for a given family member and determines their role in the family tree with a yes or no question.

II. KNOWLEDGE BASE

The knowledge base developed for this project can be divided into their specific purpose. The data in the knowledge database stores information that establishes relationships in the family tree. The given rules can be separated into three categories: 1.) Establishment of the Nuclear Family, 2.) Establishment of the Familial Roles, and 3.) Establishment of Gender roles.

<table>
<tr><th>PROLOG code</th><th>Equivalent First-Order Logic</th></tr>
<tr><td colspan="2">Establishment of the Nuclear Family</td></tr>
<tr><td>

```
% Declare dynamic predicates
so facts can be added
dynamically
:- dynamic(is_female/1).
:- dynamic(is_male/1).
:- dynamic(parent_of/2).
:- dynamic(child_of/2).
:- dynamic(son_of/2).
:- dynamic(daughter_of/2).
:- dynamic(mother_of/2).
:- dynamic(father_of/2).
:- dynamic(siblings_of/2).
:- dynamic(brother_of/2).
:- dynamic(sister_of/2).
:-
dynamic(grandfather_of/2).
:-
dynamic(grandmother_of/2).
:-
dynamic(granddaughter_of/2).
:- dynamic(grandson_of/2).
:- dynamic(grandchild/2).
:- dynamic(aunt_of/2).
:- dynamic(uncle_of/2).
:- dynamic(niece_of/2).
```

</td><td>Each line in this dynamic predicate block can be converted to the following in First-Order Logic
- is_female(X)
- is_male(X)
- parent_of (X,Y)
- child(X,Y)⇔parent(Y,X)
- son_of(X,Y)⇔is_male(X)∧child_of(X,Y)
- daughter(X,Y)⇔is_female(X)∧child_of(X,)
- mother_of(X,Y)⇔is_female(X)∧parent_of(X, Y)
- father_of(X,Y)⇔is_male(X)∧parent_of(X,Y)
- siblings_of(X,Y)⇔∃P(parent(_ofP,X)∧parent_ of(P,Y))
- brother_of(X,Y)⇔is_male(X)∧sibling_of(X,Y)
- sister_of(X,Y)⇔is_female(X)∧sibliing_of(X,Y)

Dynamic predicates allow for setting multiple properties in multiple predicates in a single call (SWI-Prolog.org) [1] This allows the program to determine the relationship of two entities simultaneously. For example, setting X as the father of Y automatically assumes that X is male and is a parent of Y.</td></tr>
</table>

```
:- dynamic(nephew_of/2).
```

```
* *** Parents Relationship *** */

/* If both are the mother and father of Z */
parents_of(X, Y, Z) :-
    parent_of(X, Z),
    parent_of(Y, Z),
    X \= Y.

/* If two siblings share the same mother/father */
parents_of(X, Y, Z) :-
    parent_of(X, Z),
    parent_of(Y, A),
    are_siblings(Z, A),
    X \= Y.
```

```
/* *** Mother Relationship *** */

/* If X is a parent of Y and X is female */
mother_of(X, Y) :-
    parent_of(X, Y),
    is_female(X).

/* If Y has a sibling that is a child of X and X is female */
mother_of(X, Y) :-
    are_siblings(Y, A),
    parent_of(X, A),
    is_female(X).
```

```
/* *** Father Relationship *** */

/* If X is a parent of Y and X is male */
father_of(X, Y) :-
    parent_of(X, Y),
    is_male(X).

/* If Y has a sibling that is a child of X and X is male */
father_of(X, Y) :-
```

```
    are_siblings(Y, A),
    parent_of(X, A),
    is_male(X).
```

```
/* If all are children of A */
children_of(X, Y, Z, A) :-
    child_of(X, A),
    child_of(Y, A),
    child_of(Z, A).
```

```
/* *** Daughter Relationship *** */

/* If X is a child of Y and X is female */
daughter_of(X, Y) :-
    child_of(X, Y),
    is_female(X).


/* *** Son Relationship *** */

/* If X is a child of Y and X is male */
son_of(X, Y) :-
    child_of(X, Y),
    is_male(X).
```

```
/* *** Sibling Relationship *** */

/* If they both have the same parents */
are_siblings(X, Y) :-
    parent_of(A, X),
    parent_of(A, Y).

/* *** Sister Relationship *** */

/* If they are siblings and X is female */
sister_of(X, Y) :-
    are_siblings(X, Y),
```

```
    is_female(X).


/* *** Brother Relationship *** */

/* If they are siblings and X is male */
brother_of(X, Y) :-
    are_siblings(X, Y),
    is_male(X).
```

PROLOG code	Equivalent First-Order Logic
Establishment of Extended Family	

`are_siblings(X, Y) :-`
`  parent_of(P, X),`
`  parent_of(P, Y),`
`  X \= Y.`
 	
`are_siblings(X, Y) :-`
`  brother_of(X,Y);`
`  sister_of(X,Y).`
	sibling_of(X,Y)⇔∃P(parent_of(P,X)∧parent_of(P,Y)∧X\=Y)
	
	Establishes that if P is a parent of X, P is also a parent of Y and X is not equal to Y then X and Y are Siblings
	
	sibling_of(X,Y)⇔(brother_of(X,Y)∨sister_of(X,Y))
	
	Establishes that X and Y are siblings if X is a brother of Y or X is a sister of Y

`mother_of(X, Y) :-`
`  parent_of(X, Y),`
`  is_female(X).`
 	
`father_of(X, Y) :-`
`  parent_of(X, Y),`
`  is_male(X).`
	mother_of(X,Y)⇔parent_of(X,Y)∧is_female(X)
	
	Establishes that X is a mother of Y if X is a parent of Y and is female
	
	father(X,Y)⇔parent_of(X,Y)∧is_male(X)
	
	Establishes that X is a father of Y if X is a parent of Y and is male

`grandfather_of(X, Y) :-`	grandfather_of(X,Y)⇔∃Z(parent_of(X,Z)∧parent_of(

parent_of(X, Z), parent_of(Z, Y), is_male(X).	Z,Y) ∧ is_male(X)) Establishes that X is a and is grandfather of Y if X is a parent of Z, Z is a parent of Y, and X is a male
grandfather_of(X, Y) :- parent_of(X, Z), parent_of(Z, Y), is_female(X).	grandmother_of(X,Y)⇔∃Z(parent_of(X,Z)∧parent_of(Z,Y) ∧ is_female(X)) Establishes that X is a and is grandmother of Y if X is a parent of Z, Z is a parent of Y, and X is a female
grandson_of(X, Y) :- grandparent_of(Y, X).	grandson(X,Y)⇔grandparent_of(Y,X)∧is_male(X)
granddaughter_of(X, Y) :- grandparent_of(Y, X).	granddaughter(X,Y)⇔grandparent_of(Y,X)∧is_female(X)
aunt_of(X, Y) :- siblings_of(X, P), parent_of(P, Y).	aunt(X,Y)⇔∃P(sibling_of(X,P)∧parent_of(P,Y)∧is_female(X))
uncle_of(X, Y) :- siblings_of(X, P), parent_of(P, Y).	uncle(X,Y)⇔∃P(sibling_of(X,P)∧parent_of(P,Y)∧is_male(X))
niece_of(X, Y) :- aunt_uncle_of(Y, X). nephew_of(X, Y) :- aunt_uncle_of(Y, X).	niece(X,Y)⇔aunt_of(Y,X)∨uncle(Y,X)∧is_female(X) nephew)of(X,Y)⇔aunt_of(Y,X)∨uncle_of(Y,X)∧is_male(X)

```
/* ***** Extended Family Establishment ***** */

/* *** Grandparent Relationship *** */

/* If X is a parent of A who is a parent of Y */
grandparent_of(X, Y) :-
    parent_of(X, A),
    parent_of(A, Y).
```

```
/* If Y has an aunt or uncle that has a parent */
grandparent_of(X, Y) :-
    auncle_of(A, Y),
    parent_of(X, A).


/* *** Grandmother Relationship *** */

/* If X is a grandparent of Y and X is female */
grandmother_of(X, Y) :-
    grandparent_of(X, Y),
    is_female(X).


/* *** Grandfather Relationship *** */

/* If X is a grandparent of Y and X is male */
grandfather_of(X, Y) :-
    grandparent_of(X, Y),
    is_male(X).


/* *** Aunt Relationship *** */

/* If X is an auncle and X is female */
aunt_of(X, Y) :-
    auncle_of(X, Y),
    is_female(X).

/* If the parent of Y has a sibling X and X is female */
aunt_of(X, Y) :-
    parent_of(A, Y),
    are_siblings(X, A),
    is_female(X).

/* If the grandparent of Y has a child X that is not the parent of Y and X is
female */
aunt_of(X, Y) :-
```

```
    grandparent_of(A, Y),
    parent_of(A, X),
    not(parent_of(X, Y)),
    is_female(X).


/* *** Uncle Relationship *** */

/* If X is an auncle and X is female */
uncle_of(X, Y) :-
    auncle_of(X, Y),
    is_male(X).

/* If the parent of Y has a sibling X and X is male */
uncle_of(X, Y) :-
    parent_of(A, Y),
    are_siblings(X, A),
    is_male(X).

/* If the grandparent of Y has a child X that is not the parent of Y and X is
male */
uncle_of(X, Y) :-
    grandparent_of(A, Y),
    parent_of(A, X),
    not(parent_of(X, Y)),
    is_male(X).
```

PROLOG code	Equivalent First-Order Logic
Establishment of Gender Roles	
`is_female(X) :-` `  mother_of(X, Y).`	mother_of(X) →is_female(X) Establishes that X is a female if X is the mother of Y

`is_female(X) :-` `  daughter_of(X, Y).`	daughter_of(X) →is_female(X) Establishes that X is a female if X is the daughter of Y
`is_female(X) :-` `  sister_of(X, Y).`	sister_of(X) →is_female(X) Establishes that X is a female if X is the sister of Y
`is_female(X) :-` `  grandmother_of(X, Y).`	grandmother_of(X) →is_female(X) Establishes that X is a female if X is the grandmother of Y
`is_female(X) :-` `  aunt_of(X, Y).`	aunt_of(X) →is_female(X) Establishes that X is a female id X is the aunt of Y
`is_male(X) :-` `  father_of(X, Y).`	father_of(X) →is_male(X) Establishes that X is male if X is the father of Y
`is_male(X) :-` `  son_of(X, Y).`	son_of(X) →is_male(X) Establishes that X is male if Y is the son of X
`is_male(X) :-` `  brother_of(X, Y).`	brother_of(X) →is_male(X) Establishes that X is male if X is the brother of Y
`is_male(X) :-` `  grandfather_of(X, Y).`	grandfather_of(X) →is_male(X) Establishes that C is male if X is the father Y
`is_male(X) :-` `  uncle_of(X, Y).`	uncle_of(X) →is_male(X,Y) Establishes that C is male if X is the uncle of Y

III. CHATBOT IMPLEMENTATION

This chatbot is developed using PROLOG's logic-based environment to develop a knowledge base that is comprised of relationship rules created based on family relation naming convention in english. The user interface is added using python and primarily uses the console to ask for prompts. User input will then be parsed and translated into the proper Prolog queries that will reference the database. The prompt should follow a specific sentence pattern as per the requirements of the project specification. Handling of incorrect prompts should be delt by the program. Furthermore, incomplete facts should also lead to a message that indicates incomplete information. This program should have the basic capabilities of determining familial relationships based on given facts.

Question prompts always end with a question mark (?) and can be one of the following sentence patterns:

Sentence Pattern	
Are [ ] and [ ] siblings?	Who are the siblings of [ ]?
Is [ ] a sister of [ ]?	Who are the sisters of [ ]?
Is [ ] a brother of [ ]?	Who are the brothers of [ ]?
Is [ ] the mother of [ ]?	Who is the mother of [ ]?
Is [ ] the father of [ ]?	Who is the father of [ ]?
Are [ ] and [ ] the parents of [ ]?	Who are the parents of [ ]?
Is [ ] a grandmother of [ ]?	Is [ ] a grandfather of [ ]?
Is [ ] a daughter of [ ]?	Who are the daughters of [ ]?
Is [ ] a son of [ ]?	Who are the sons of [ ]?
Is [ ] a child of [ ]?	Who are the children of [ ]?
Are [ ], [ ] and [ ] children of [ ]?	Is [ ] an aunt of [ ]?
Is [ ] an uncle of [ ]?	Are [ ] and [ ] relatives?

The chatbot should respond to a question prompt by providing the appropriate answer. For yes/no questions, the chatbot should answer either "yes", if the knowledge base can prove it, or "no", if it cannot.

Figure 3.1. Expected sentence pattern of question prompts

IV. RESULTS

Using the chatbot that was made, where the Python script integrates PROLOG, the results show that it is functional. The given code successfully adds facts (e.g. parents_of, is_female, is_male) via Python into the PROLOG knowledge base. It also queries the respected relationship (e.g. sibling_of,

grandparent_of). The system handles these queries properly since it shows the correct and proper inference of relationships from the given facts.

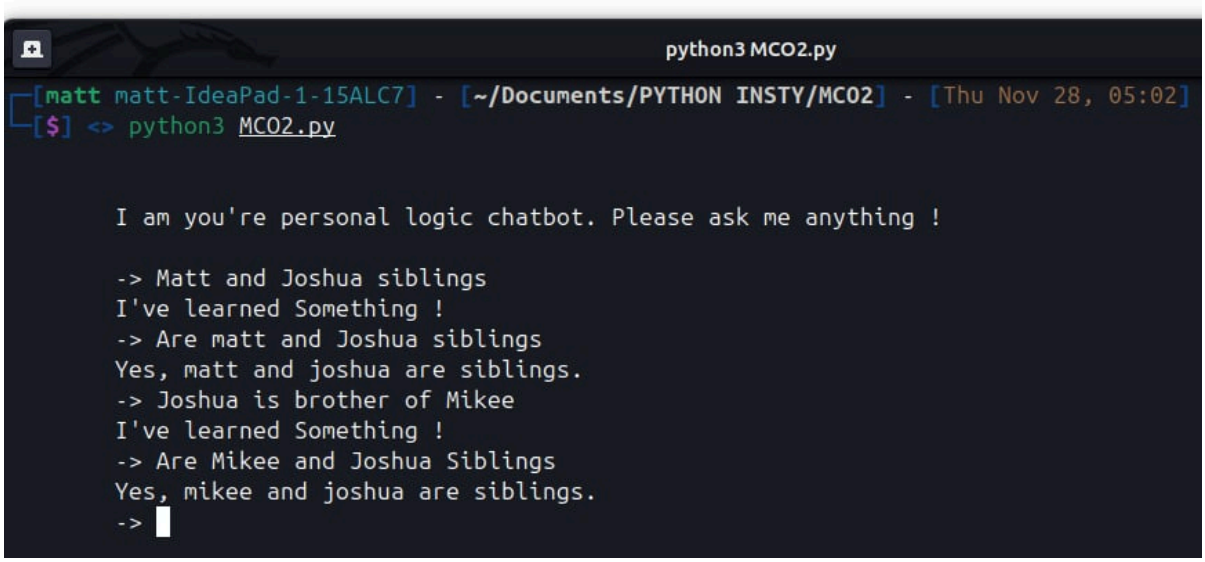

Figure 4.1. Sample Run 1 of Chatbot

We can see in Figure 4.1. that the user has inputted new objects, namely Matt and Joshua, and these two objects are then inputted as siblings. The previous statement is now a fact in the knowledge base. When asked if Matt and Joshua are siblings, the system responded that they are since it is a fact. The user then inputs a new object, Mikee, to be the brother of Joshua. Since it is established in the knowledge base that if an object is a brother of another object, then they are siblings. Thus when asking Chatbot if Mikee and Joshua are siblings, it says that they are.

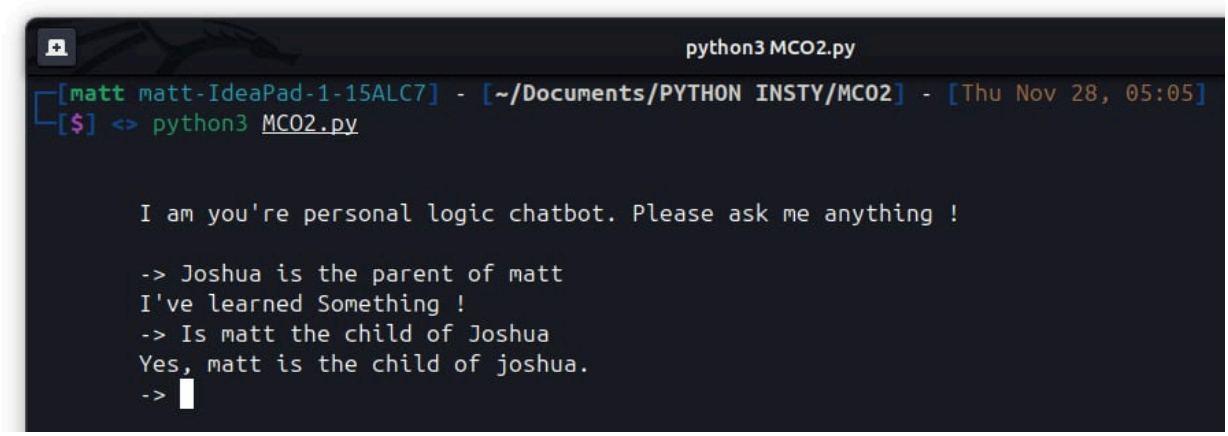

Figure 4.2. Sample Run 2 of Chatbot

We can see in Figure 4.2 that the user has inputted new objects, namely Matt and Joshua, where Joshua is the parent of Matt. The previous statement is now a fact in the knowledge base. Since it is established in the database that if object X is a parent of object Y, then object Y is the child of object X. Thus when asking Matt is the child of Joshua, it agrees and says that he is.

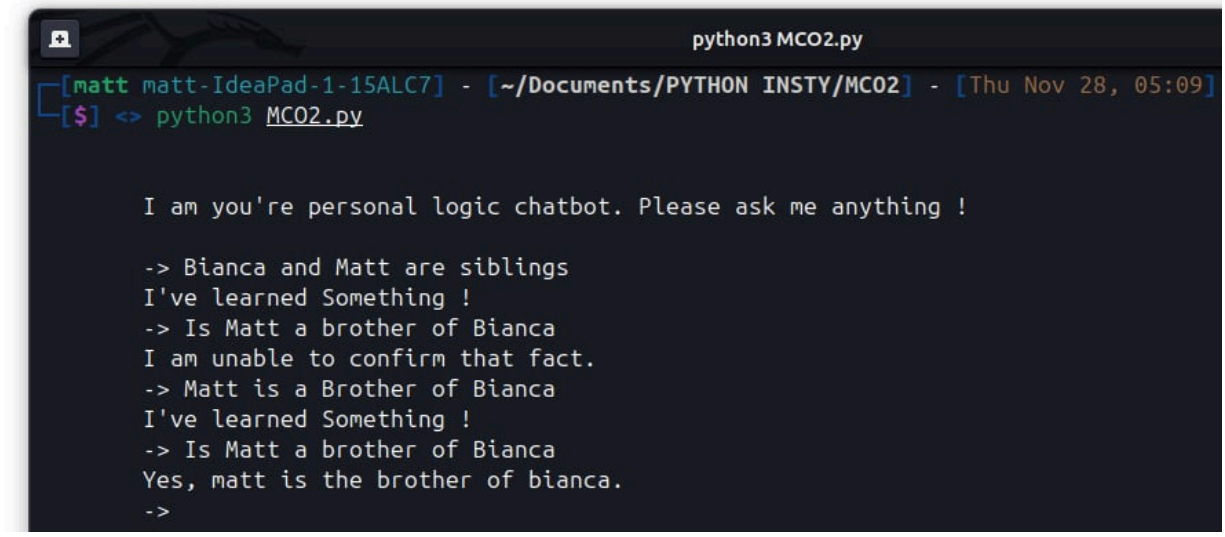

Figure 4.3. Sample Run 3 of Chatbot

We can see in Figure 4.3. that the user has inputted new objects, namely Bianca and Matt, and these two objects are then inputted as siblings. The previous statement is now a fact in the knowledge base. When asked if Matt is the brother of

Bianca, it responds with “I am unable to confirm that fact” since if an object is a sibling of another object, then it is possible that they could be a brother or a sister. Since the chatbot has limited inferences, then the relationship of one fact to another is only one-to-one. The user then inputs a new fact, Matt is now a brother of Bianca. Since the previous statement is established in the knowledge base, when asking Chatbot if Matt is the brother of Bianca, it agrees and says that he is.

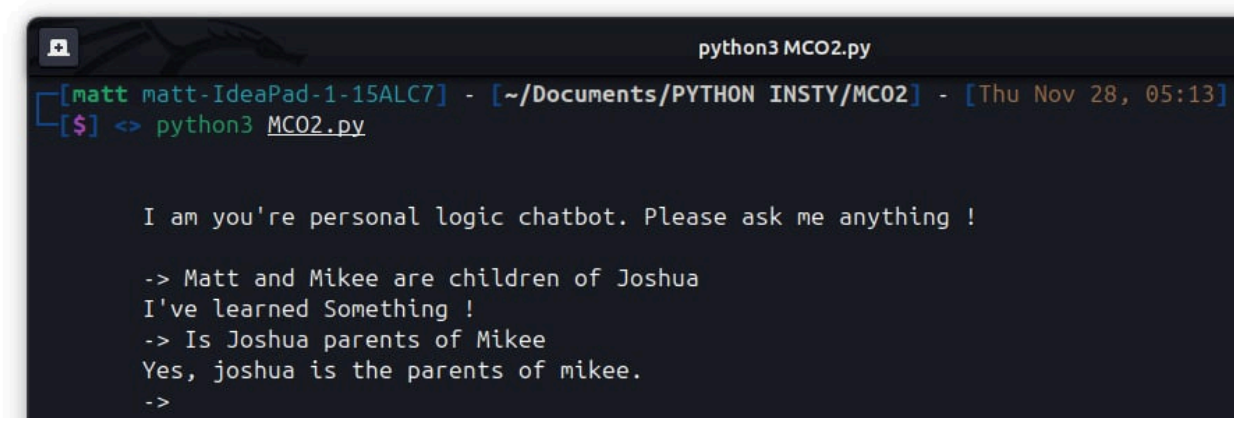

Figure 4.4. Sample Run 4 of Chatbot

We can see in Figure 4.4. that the user has inputted new objects, namely Matt and Mikee, to be the children of Joshua. The previous statement is now a fact in the knowledge base. Since it is established in the knowledge base that if objects X and Y are the children of object Z, then object Z is the parent of either object X or Y. Thus when asking Chatbot if Joshua is the parent of Mikee, it agrees and says that he is.

V. LIMITATIONS AND CHALLENGES

One of the challenges faced when making the Chatbot is that the system is dependent on PROLOG. It needs to be properly installed and the “Social_Rules.pl” needs to be correctly located for Chatbot to work. Another requirement is that the psywip library is also required. This may not be compatible with all versions of Python and PROLOG, which can result in unexpected crashes and problems.

Chatbot is also limited to its scalability. As the knowledge base is filled with more information, querying complex relationships may become slower. Another is that there are certain complex relationships that depend on negation or multi-level logic. This could result in incorrect results if the knowledge base is incomplete.

In comparison to other Large Language Models (LLM) (e.g. ChatGPT, Copilot, ClaudeAI), Chatbot is only limited to the information available to its knowledge base, while LLMs are not only limited to their knowledge base but their information can also come from the Internet. LLMs handle large amounts of data and complex queries without slowing down, but with PROLOG, it may experience performance issues such as bottlenecks if there are large quantities of data involved. Another is the adaptability of LLMs, where they can understand abstract and ambiguous inputs, while PROLOG requires you to be specific in your input. Lastly, LLMs can handle inconsistent data by filling in the gaps by its reasoning, while PROLOG can not since its reasoning is based on its knowledge base only, thus limiting its ability to adapt.

VI. CONCLUSION

Although there may be limitations when it comes to our Chatbot in comparison to its LLM counterparts and its own requirements, it proved to be successful based on our results. It can handle accepting new facts and can query relationships properly. The knowledge base built with PROLOG can be read and understood by Chatbot and can be used for a proper output for a respective input.

Chatbot's outputs are also accurate, provided that the required facts and rules are present. Additionally, Chatbot's potential to evolve is seen since it can receive new facts and add them to its knowledge base. These new facts can then open opportunities for customization and adaptability in specific domains.

In conclusion, it showcases the potential of a PROLOG-based Chatbot. It is also precise and reliable, given the data constraints in its knowledge base. Overall, Chatbot proved to be a success and this project is proof that it is so.

VII. APPENDIX AND BIBLIOGRAPHY

[1] SWI-Prolog -- (dynamic)/2

VIII. TABLE OF CONTRIBUTIONS

MEMBER	CONTRIBUTION
Copon, Matthew Joshua	• Led the UI implementation for the ChatBot and constructed the main method for ChatBot to run. • Actively contributed to the research, brainstorming, discussions, and problem-solving for the MCO
Fabiaña, Joshua Jose	• Contributed written content and played an active role in developing the ChatBot, focusing on both theoretical and practical modeling approaches. • Engaged in research, brainstorming, and collaborative discussions essential to the MCO's progress.
Kelsey, Gabrielle Madison	• Played a significant role in building the knowledge base for the ChatBot using PROLOG, contributing

	significantly to refining the aspects of the group's ChatBot. ● Engaged deeply in research, brainstorming, and problem-solving to enhance the project's knowledge base and results.
Namuag, Leigh Avegail	● Provided writing contributions and was actively involved in the construction of the ChatBot, contributing to theoretical and practical modeling strategies. ● Participated in research, brainstorming, and discussions critical to the MCO's development.
Perdon, Linus Carl	● Played a key role in writing and actively contributed to the development of the ChatBot, incorporating both theoretical and practical modeling techniques. ● Took part in research, brainstorming, and discussions vital to the advancement of the MCO.