AI CHATBOT for Agriculture IN RASA

A Project Report

Submitted in Partial Fulfillment of the Requirement for the Award of the Degree

of

BACHELOR OF TECHNOLOGY

(Computer Science and Engineering)

To

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY

Dharwad

ज्ञानेन विकासः

By

Kuluru Vineeth Kumar Reddy	**REG NO:18BCS043**
Karthick P S	**REG NO:18BCS038**
LaxmiNarayana K	**REG NO:18BCS037**

Under the Guidance of

Dr. Uma Seshadri

DEPARTMENT OF COMPUTER SCIENCE

IIIT, Dharwad

APR-2021

CANDIDATE'S DECLARATION

We hereby certify that the work which is being presented in the project report entitled "**AI CHATBOT for Agriculture IN RASA**" in partial fulfillment of the requirement for the award of the Degree of Bachelor of Technology and submitted to the Department of Computer Science of Indian Institute of Information Technology Dharwad, is an authentic record of our own work carried out during a period from February 2021 to April 2021 under the supervision of Dr. Uma Seshadri, Department of Computer Science and Technology, Indian Institute of Information Technology, Dharwad.

The matter presented in this report has not been submitted by us for the award of any other degree of this or any other Institute/University.

Kuluru Vineeth Kumar Reddy

Karthick P S

Laxminarayana K

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: Dr. Uma Seshadri

ACKNOWLEDGEMENTS

It is indeed a great pleasure to express our sincere thanks to our supervisor Dr. Uma Seshadri, Department of Computer Science and Engineering, IIIT Dharwad for her continuous support in this project. She was always there to listen, advise and share her expertise with us at every stage of the project development . She showed us different ways to approach a real world problem and the need to be persistent to accomplish any goal. She had confidence in us when we had doubted ourselves, and brought out the good ideas in us. She was always there to meet and talk about our ideas, share her expertise and views on developing a solid prototype, and to ask us good questions to help us think through our problems. Without her encouragement and constant guidance, we could not have completed this project on time.

We are also indebted to our own team members for their rigorous efforts in questioning the most difficult edge cases and extracting the best out of it and also for their persistent coordination till the end. Without their support and cooperation, this project could not have been finished.

Kuluru Vineeth Kumar Reddy

Karthick P S

Laxminarayana K

CONTENTS

Phase-1

INTRODUCTION

1.1 ABSTRACT

Chatbots are computer programs that simulate human conversation through voice commands or text chats or both. Chatbot, short for chatterbot, is an artificial intelligence (AI) feature that can be embedded and used through any major messaging applications.

The goal of this project is to build a prototype of an AI chatbot to address the problems faced by farmers in various phases of the agricultural sector.

Our AI chatbot has features ranging from providing the information of fertilizer consumption state wise, educating the farmers about the MSP rates in their respective states(for a limited crop varieties), acknowledging them to use specific fertilizer varieties to reduce their initial investment and suggest for proper crops to be grown based on the features like nutrient content of soil after having gone through soil testing(ex:N,P,K values of soil),live weather details of their residing location(ex: temperature, humidity, rainfall). The chatbot also provides these recommendations to the farmer in his native language(for demonstration we have used kannada).

Our chatbot is built exclusively using RASA, an open source Machine Learning framework which uses the RASA NLU for understanding the user intents and RASA Core to predict the best possible action as a response from the chatbot based on a probabilistic model.

The chatbot uses the publicly available APIs from various government portals and publicly available datasets from open source communities like Kaggle to access the information required. The project also makes use of techniques such as input feature extraction from the raw data collected ,which is feeded as an input to the predictive Machine Learning model.It also uses the different datasets from these portals to make use of them for **training an ML model for crop recommendation.**

1.2 AI and NLP

Natural Language Processing (NLP) refers to an AI method of communicating with intelligent systems using a natural language such as English.

Processing of Natural Language is required when you want an intelligent system like a robot to perform as per your instructions, when you want to hear decisions from a dialogue based clinical expert system, etc.

The field of NLP involves making computers to perform useful tasks with the natural languages humans use. The input and output of an NLP system can be −

- Speech
- Written Text

Components of NLP

There are two components of NLP as given −

- Natural Language Understanding (NLU)
  Understanding involves the following tasks −
  1. Mapping the given input in natural language into useful representations.
  2. Analyzing different aspects of the language.
- Natural Language Generation (NLG)
  It is the process of producing meaningful phrases and sentences in the form of natural language from some internal representation. It involves −
  1. Text planning − It includes retrieving the relevant content from a knowledge base.
  2. Sentence planning − It includes choosing required words, forming meaningful phrases, setting the tone of the sentence.
  3. Text Realization − It is mapping sentence plan into sentence structure.

Difficulties in NLU:

NL has an extremely rich form and structure.

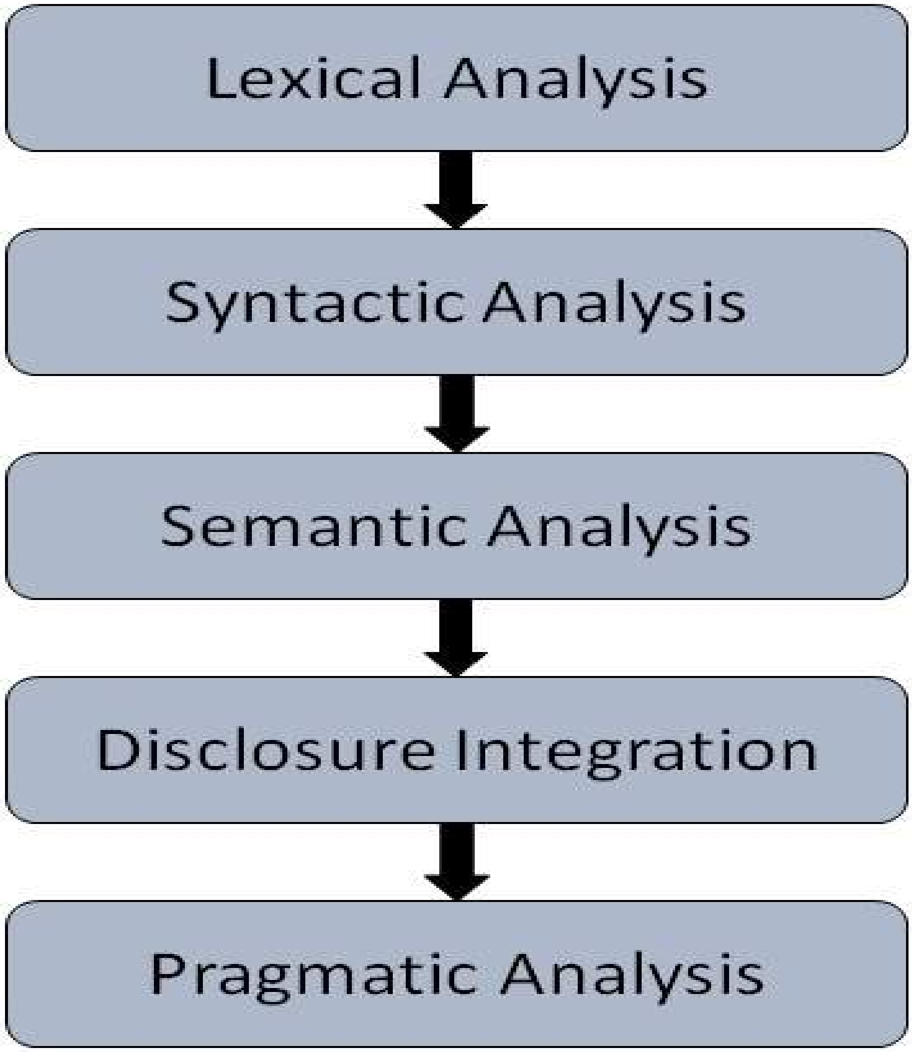
It is very ambiguous. There can be different levels of ambiguity −

- Lexical ambiguity − It is at very primitive level such as word-level.
- For example, treating the word "board" as noun or verb?
- Syntax Level ambiguity − A sentence can be parsed in different ways.
- For example, "He lifted the beetle with red cap." − Did he use cap to lift the beetle or he lifted a beetle that had red cap?
- Referential ambiguity − Referring to something using pronouns. For example, Rima went to Gauri. She said, "I am tired." − Exactly who is tired?
- One input can mean different meanings.
- Many inputs can mean the same thing.

Steps in NLP:

There are general five steps −

- ***Lexical Analysis*** − It involves identifying and analyzing the structure of words. Lexicon of a language means the collection of words and phrases in a language. Lexical analysis is dividing the whole chunk of text into paragraphs, sentences, and words.
- ***Syntactic Analysis (Parsing)*** − It involves analysis of words in the sentence for grammar and arranging words in a manner that shows the relationship among the words. The sentence such as "The school goes to boy" is rejected by English syntactic analyzers.



- ***Semantic Analysis*** − It draws the exact meaning or the dictionary meaning from the text. The text is checked for meaningfulness. It is done by mapping syntactic structures and objects in the task domain. The semantic analyzer disregards sentences such as "hot ice-cream".
- ***Discourse Integration*** − The meaning of any sentence depends upon the meaning of the sentence just before it. In addition, it also brings about the meaning of immediately succeeding sentences.

- ***Pragmatic Analysis*** − During this, what was said is re-interpreted on what it actually meant. It involves deriving those aspects of language which require real world knowledge.

1.3 Natural Language Understanding(NLU)

Natural language understanding (NLU) is a branch of natural language processing (NLP), which involves transforming human language into a machine-readable format. With the help of natural language understanding (NLU) and machine learning, computers can automatically analyze data in seconds, saving businesses countless hours and resources when analyzing troves of customer feedback.

NLU vs NLP:

- Natural language understanding is a subfield of natural language processing.
- Both NLP and NLU aim to make sense of unstructured data, but there is a difference between the two.
- NLP is concerned with how computers are programmed to process language and facilitate “natural” back-and-forth communication between computers and humans.
- Natural language understanding, on the other hand, focuses on a machine’s ability to understand the human language. NLU refers to how unstructured data is rearranged so that machines may “understand” and analyze it.

Let us look at it this way. Before a computer can process unstructured text into a machine-readable format, first machines need to understand the peculiarities of the human language. So NLU plays a very important role in helping our chatbot to understand unstructured data and make use of them to actively participate with the users to drive the conversation.

ex.

Machine Translation (MT):

Accurately translating text or speech from one language to another is one of the toughest challenges of natural language processing and natural language understanding.Using complex algorithms that rely on linguistic rules and AI machine training, Google Translate, Microsoft Translator, and Facebook Translation have become leaders in the field of “generic” language translation.

Phase-2

RASA and RASA X

2.1 Introduction to RASA

What are contextual assistants?

- Able to understand the context of the conversation, i.e. what the user has said previously and when/where/how they said it.
- Capable of understanding and responding to different and unexpected inputs.
- Can learn from previous conversations and improve in accuracy over time Buildable today with Rasa.

Exploring Rasa

Rasa has three major components that work together to create contextual assistants:

Rasa NLU:

Rasa NLU is like the "ear" of your assistant—it helps your assistant understand what's being said. Rasa NLU takes user input in the form of unstructured human language and extracts structured data in the form of intents and entities.

- Intents are labels that represent the goal, or meaning, of a user's specific input. For example, the message 'Hello' could have the label 'greet' because the meaning of this message is a greeting.
- Entities are important keywords that an assistant should take note of. For example, the message 'My name is Juste' has the name 'Juste' in it. An assistant should extract the name and remember it throughout the conversation to keep the interaction natural.

    1. Entity extraction is achieved by training a named entity recognition model to identify and extract the entities (in this example, names) for unstructured user messages

Rasa Core:

Core is Rasa's dialogue management component. It decides how an assistant should respond based on:

1) The state of the conversation and

2) The context. Rasa Core learns by observing patterns in conversational data between users and an assistant.

Rasa X:

Rasa X is a toolset for developers to build, improve and deploy contextual assistants with the Rasa framework. You can use Rasa X to:

-View and annotate conversations

-Get feedback from testers

-Version and manage models

With Rasa X, you can share your assistant with real users and collect the conversations they have with the assistant, allowing you to improve your assistant without interrupting the assistant running in production

2.2 Generating the NLU training data(intents and entities)

The moodbot starter project contains a Data directory, where we will be able to find the training data files for NLU and dialogue management models. The Data directory contains two files:

- **nlu.md** - the file containing NLU model training examples. This includes intents, which are user goals, and example utterances that represent those intents. The NLU training data also labels the entities, or important keywords, the assistant should extract from the example utterance.
    1. Intents are defined using a double hashtag. Each intent is followed by multiple examples of how a user might express that intent.

```
intent: recommend
examples: |
  - crop_recommendation
  - recommendation of crop
intent: statistics
examples: |
  - crop statistics
  - /statistics
intent: about_us
examples: |
  - we are agri helpers
  - agri helpers
  - agro helpers
intent: agribot
examples: |
  - /agribot
  - chat with agribot
  - go with agribot
  - agribot
intent: nitrogen_entry
examples: |
  - [N_10](nitrogen)
  - [N_20](nitrogen)
  - [N_30](nitrogen)
  - [N_40](nitrogen)
```

2. Entities are labeled with square brackets and tagged with their type in parentheses.(screenshot of entities by laxy)

```
examples: |
    - [carrot](crop)
    - [potato](crop)
    - [tomato](crop)
    - [cucumber](crop)
    - [Drumstick](crop)
    - [cotton](crop)
    - [rice](crop)
    - [jute](crop)
    - [chickpea](crop)
    - [kidneybeans](crop)
    - [pigeonpeas](crop)
    - [mothbeans](crop)
    - [mungbean](crop)
    - [blackgram](crop)
    - [lentil](crop)
    - [pomegranate](crop)
    - [banana](crop)
    - [mango](crop)
    - [grapes](crop)
    - [watermelon](crop)
    - [muskmelon](crop)
    - [apple](crop)
    - [orange](crop)
    - [papaya](crop)
    - [coconut](crop)
    - [coffee](crop)
```

- **stories.md** - the file containing story data. Stories are example end-to-end conversations

```
- story: About us path
  steps:
  - intent: greet
  - action: utter_greet
  - intent: about_us
  - action: utter_about_us
- story: price path
  steps:
  - intent: greet
  - action: utter_greet
  - intent: agribot
  - action: utter_agribot
  - intent: statistics
  - action: utter_statistics
  - intent: price
  - action: utter_ask_crop1
  - intent: crop_entry
  - action: utter_tc_as
  - intent: state_entry
  - action: action_get_price
```

```
- story: fertilizers path
  steps:
  - intent: greet
  - action: utter_greet
  - intent: agribot
  - action: utter_agribot
  - intent: statistics
  - action: utter_statistics
  - intent: ferti
  - action: utter_ferti
  - intent: state_entry
  - action: action_get_fertilizers
```

Rules are a type of training data used to train your assistant's dialogue management model. Rules describe short pieces of conversations that should always follow the same path.

```yaml
rules:

- rule: Say goodbye anytime the user says goodbye
  steps:
  - intent: goodbye
  - action: utter_goodbye

- rule: Say 'I am a bot' anytime the user challenges
  steps:
  - intent: bot_challenge
  - action: utter_iamabot

- rule: Say out_of_scope
  steps:
  - intent: out_of_scope
  - action: utter_fallback
```

PRE-CONFIGURED RASA PIPELINES:

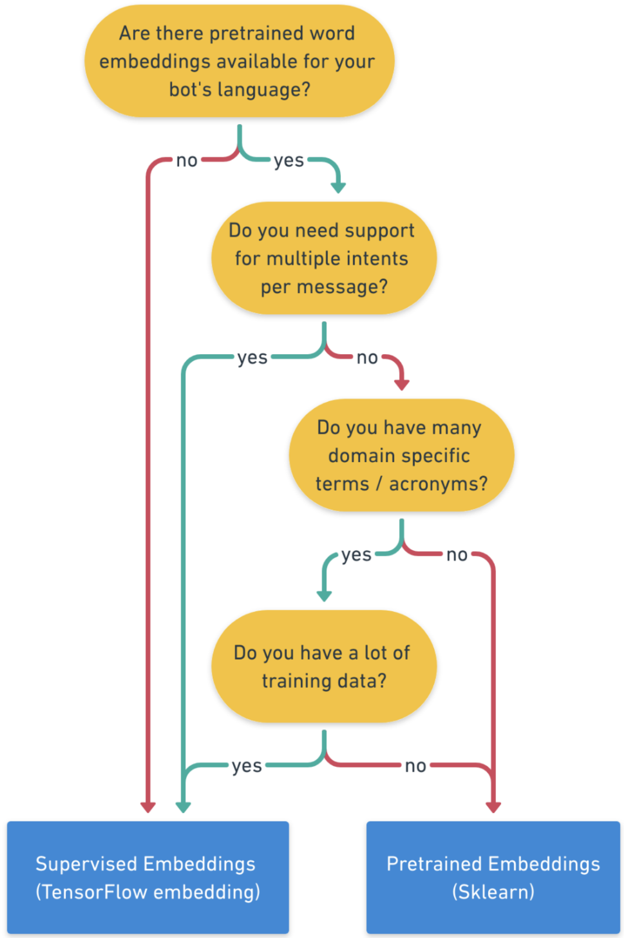
Key Concepts

- **NLU model** - An NLU model is used to extract meaning from text input.Training an NLU model on this data allows the model to make predictions about the intents and entities in new user messages, even when the message doesn’t match any of the examples the model has seen before.
- **Training pipeline** - NLU models are created by a training pipeline, also referred to as a processing pipeline. A training pipeline is a sequence of processing steps which allow the model to learn the training data’s underlying patterns.
- **Word embeddings** - Word embeddings convert words to vectors, or dense numeric representations based on multiple dimensions. Similar words are represented by similar vectors,which allows the technique to capture their meaning. Word embeddings are used by the training pipeline components to make text data understandable to the machine learning model.

Rasa comes with two default, pre-configured pipelines

1. **Pretrained_embeddings_spacy** : Uses the spaCy library to load pre-trained language models, which are used to represent each word in the user's input as word embeddings.
2. **Supervised_embeddings**: Unlike pre-trained embeddings, the supervised_embeddings pipeline trains the model from scratch using the data provided in the NLU training data file.

```yaml
pipeline:
  - name: WhitespaceTokenizer
  - name: RegexFeaturizer
  - name: LexicalSyntacticFeaturizer
  - name: CountVectorsFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: char_wb
    min_ngram: 1
    max_ngram: 4
  - name: DIETClassifier
    epochs: 100
    constrain_similarities: true
    model_confidence: softmax
  - name: EntitySynonymMapper
  - name: ResponseSelector
    epochs: 100
    constrain_similarities: true
    model_confidence: softmax
  - name: FallbackClassifier
    threshold: 0.3
    ambiguity_threshold: 0.1
```

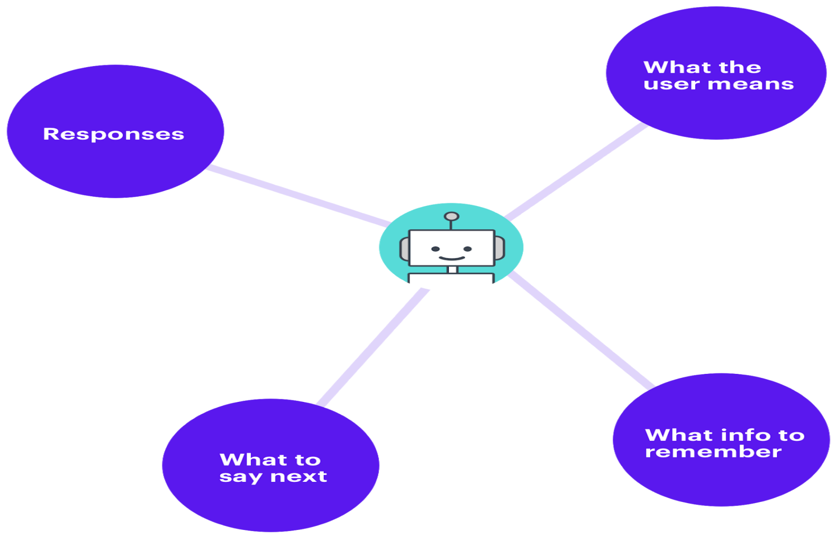


2.3 Domain,Custom actions and slots

Domain File in Rasa:

The domain is an essential component of a Rasa dialogue management model. It defines the environment in which the assistant operates, including:

- What the user means: specifically, what intents and entities the model can understand.
- What responses the model can provide: such as utterances or custom actions.
-
- What to say next: what the model should be ready to respond with.
- What info to remember: what information an assistant should remember and use throughout the conversation.




```
responses:
  utter_greet:
  - buttons:
    - payload: /about_us
      title: About us
    - payload: /agribot
      title: Chat with AGRIBOT
    text: Welcome to AGRIGROW!, All your farming needs at one place.
  utter_cheer_up:
  - image: https://i.imgur.com/nGF1K8f.jpg
    text: 'Here is something to cheer you up:'
  utter_did_that_help:
  - text: Did that help you?
  utter_happy:
  - text: Great, carry on! and will work more to assist you in all ways
  utter_goodbye:
  - text: Bye
  utter_iamabot:
  - text: I am a bot, powered by Rasa.
  utter_fallback:
  - text: Sorry,I did not get you,Please enter the correct input
  utter_greet1:
  - text: Hey i can help you know the msp of various crops.please enter your crop
  utter_tc_as:
  - text: Great! your crop is {crop} now please enter your state
  utter_ferti:
  - text: Oh yes i can give you fertilizer information statewise! please enter your state
  utter_ask_name:
  - text: May i know your name please
  utter_ask_number:
  - text: May i know your mobile number please
  utter_ask_land:
  - text: May i know land registered in your name please
  utter_ask_quantity:
  - text: May i know the quantity u want to register please
  utter_ask_aadharno:
  - text: May i know your aadhar id please
```

Actions:

The section called actions should contain the list of all utterances and custom actions an assistant should use to respond to user's inputs. These should come from your stories data in the stories.md file.

```
actions:
- action_get_price
- action_get_fertilizers
- action_ferti_recommendation
- action_crop_recommendation
```

Custom Actions in Rasa:

Adding response templates directly to the domain file is the easiest way to define the message an assistant sends the user once a specific utterance is predicted. But there is another way to achieve the same result - by creating custom actions. Custom actions are response actions which include custom code. That custom code can define anything from a simple text response to a backend integration - an API call, connecting to the database, or anything else your assistant needs to do.

Custom actions are defined in a file called **actions.py**, containing python code, as the file extension suggests

- **tracker** keeps track of what happens at each point within a dialogue - what intents were predicted, which entities where extracted, as well as other information
- **dispatcher** is the element that sends the response back to the user

  (screenshots of custom actions)

```
lass ActionFertilizerConsumption(Action):

    def name(self) -> Text:
        return "action_get_fertilizers"

    def run(self, dispatcher: CollectingDispatcher,
            tracker: Tracker,
            domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        loc = tracker.get_slot('location')
        api_key = '579b464db66ec23bdd000001cdd3946e44ce4aad7209ff7b23ac5

        current = requests.get('https://api.data.gov.in/resource/1a800a9
        print(current)

        fur=current.json()['records'][0]['urea']
        fdap=current.json()['records'][0]['dap']
        fmop=current.json()['records'][0]['mop']
        fcomplex=current.json()['records'][0]['complex']
        fssp=current.json()['records'][0]['ssp']
        ftot=current.json()['records'][0]['_total']

        response='''The urea used in {} in the year 2016-17 was {} \n Th
        translator = Translator()
        t_text1 = translator.translate(response,dest='kn')
        print(t_text1.text)
        language = 'kn'
        audio = gTTS(text=t_text1.text, lang=language, slow=False)
        audio.save("ferti_kannada.mp3")
        os.system("ferti_kannada.mp3")
        dispatcher.utter_message(text=t_text1.text)

        return []
```

Slots in Rasa:

Another important element of the domain file - very important for dialogue management in Rasa - is slots. Slots function as the assistant's memory, and are used by your assistant to remember important details throughout the conversation and apply those details in context to drive the conversation. Slots act as a key-value pair to store information critical to the conversation with the user. This information can be provided by the user (e.g., entity values extracted by the NLU model) or gathered from outside the conversation (e.g.,results extracted from the external database).

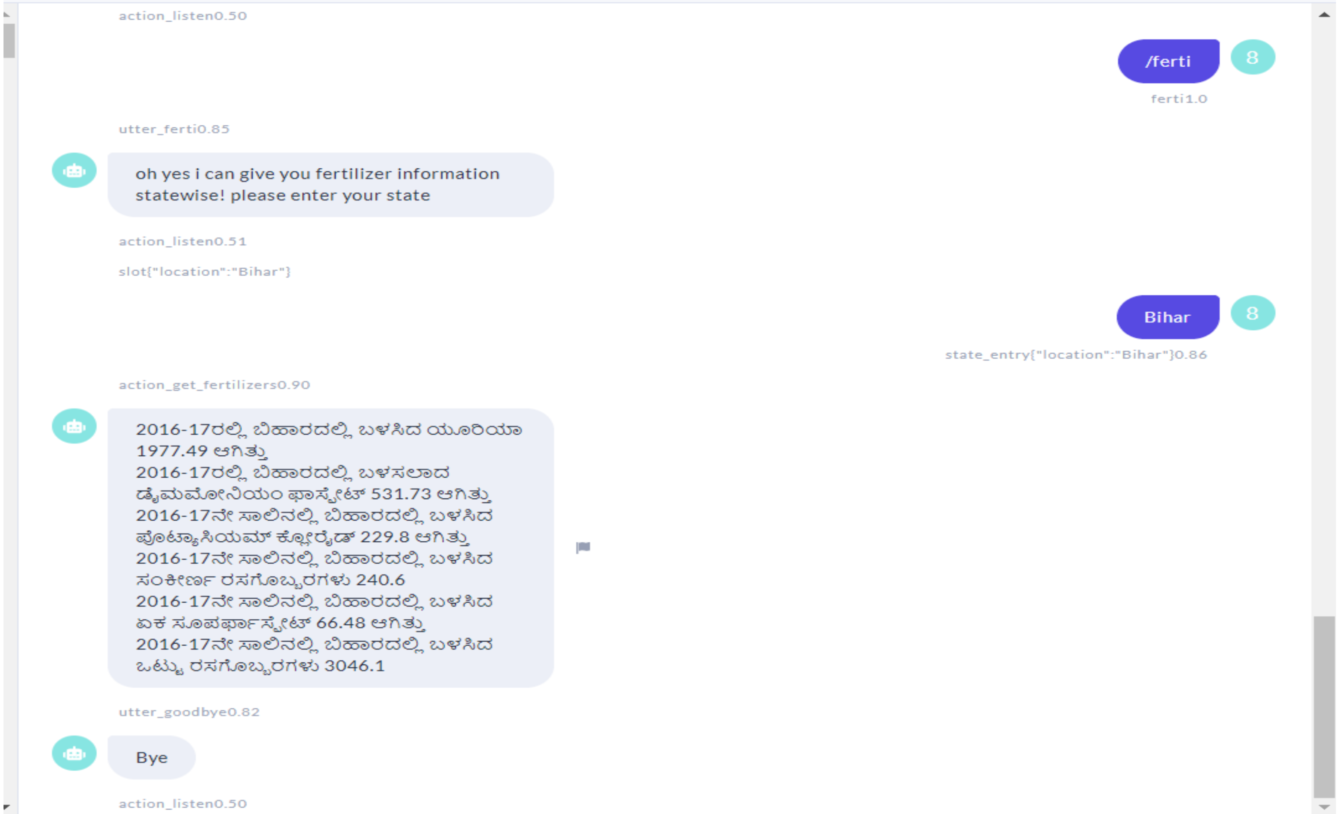
```yaml
slots:
  location:
    type: rasa.shared.core.slots.TextSlot
    initial_value: null
    auto_fill: true
    influence_conversation: true
  crop:
    type: rasa.shared.core.slots.TextSlot
    initial_value: null
    auto_fill: true
    influence_conversation: true
  nitrogen:
    type: rasa.shared.core.slots.TextSlot
    initial_value: null
    auto_fill: true
    influence_conversation: true
  phosphorous:
    type: rasa.shared.core.slots.TextSlot
    initial_value: null
    auto_fill: true
    influence_conversation: true
```

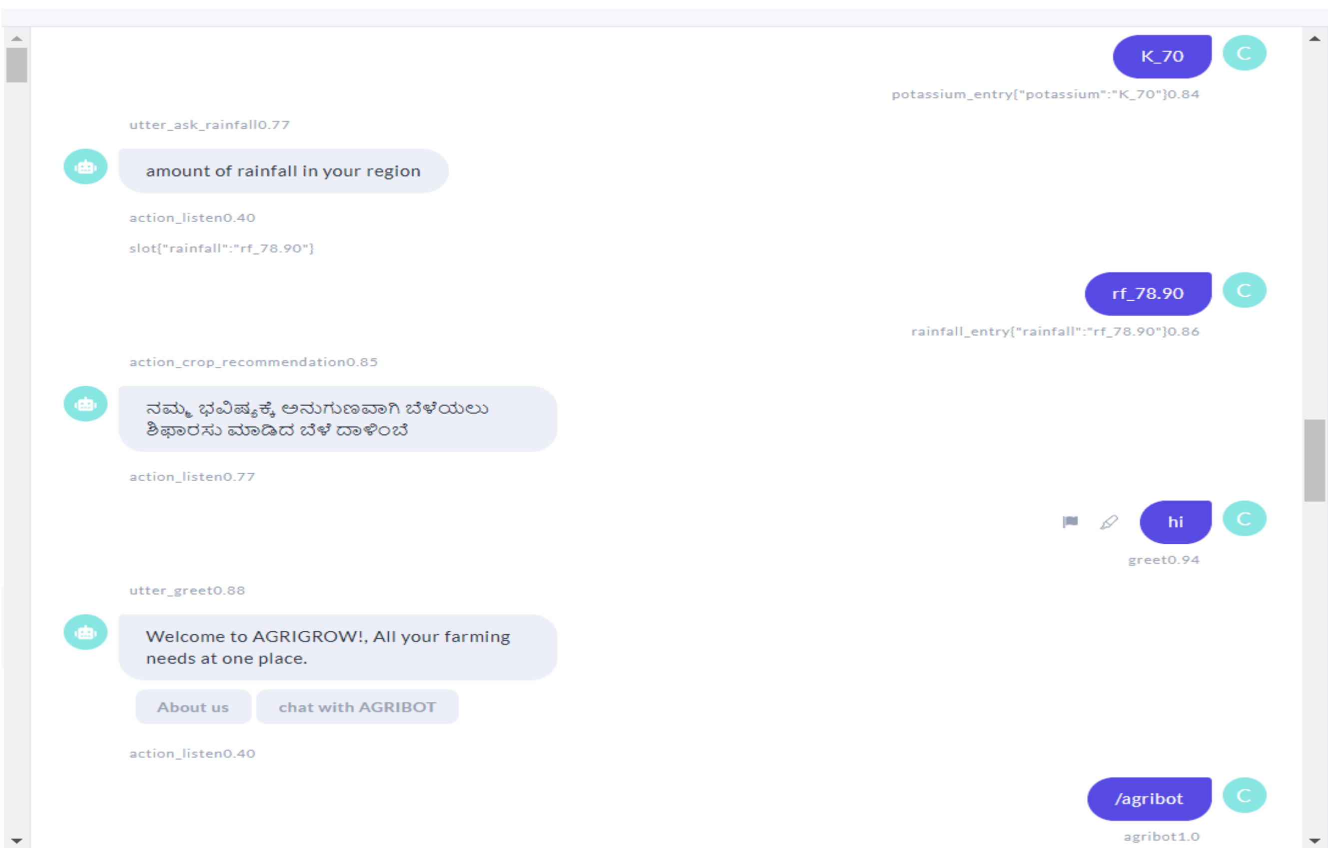
2.4: RASA X

What is Rasa X?

Rasa X is a UI tool for developers, used to improve assistants built with Rasa Open Source. It's intended to solve two problems:

- First, to make it easier to leverage real conversations as training data.
- Second, to provide a way to review past conversations for patterns or errors


action_listen0.50
/ferti
8
ferti1.0
utter_ferti0.85
oh yes i can give you fertilizer information statewise! please enter your state
action_listen0.51
slot{"location":"Bihar"}
Bihar
8
state_entry{"location":"Bihar"}0.86
action_get_fertilizers0.90
2016-17ರಲ್ಲಿ ಬಿಹಾರದಲ್ಲಿ ಬಳಸಿದ ಯೂರಿಯಾ 1977.49 ಆಗಿತ್ತು
2016-17ರಲ್ಲಿ ಬಿಹಾರದಲ್ಲಿ ಬಳಸಲಾದ ಡೈಮಮೋನಿಯಂ ಫಾಸ್ಫೇಟ್ 531.73 ಆಗಿತ್ತು
2016-17ನೇ ಸಾಲಿನಲ್ಲಿ ಬಿಹಾರದಲ್ಲಿ ಬಳಸಿದ ಪೊಟ್ಯಾಸಿಯಮ್ ಕ್ಲೋರೈಡ್ 229.8 ಆಗಿತ್ತು
2016-17ನೇ ಸಾಲಿನಲ್ಲಿ ಬಿಹಾರದಲ್ಲಿ ಬಳಸಿದ ಸಂಕೀರ್ಣ ರಸಗೊಬ್ಬರಗಳು 240.6
2016-17ನೇ ಸಾಲಿನಲ್ಲಿ ಬಿಹಾರದಲ್ಲಿ ಬಳಸಿದ ಏಕ ಸೂಪರ್ಫಾಸ್ಫೇಟ್ 66.48 ಆಗಿತ್ತು
2016-17ನೇ ಸಾಲಿನಲ್ಲಿ ಬಿಹಾರದಲ್ಲಿ ಬಳಸಿದ ಒಟ್ಟು ರಸಗೊಬ್ಬರಗಳು 3046.1
utter_goodbye0.82
Bye
action_listen0.50


K_70
C
potassium_entry{"potassium":"K_70"}0.84
utter_ask_rainfall0.77
amount of rainfall in your region
action_listen0.40
slot{"rainfall":"rf_78.90"}
rf_78.90
C
rainfall_entry{"rainfall":"rf_78.90"}0.86
action_crop_recommendation0.85
ನಮ್ಮ ಭವಿಷ್ಯಕ್ಕೆ ಅನುಗುಣವಾಗಿ ಬೆಳೆಯಲು ಶಿಫಾರಸು ಮಾಡಿದ ಬೆಳೆ ದಾಳಿಂಬೆ
action_listen0.77
hi
C
greet0.94
utter_greet0.88
Welcome to AGRIGROW!, All your farming needs at one place.
About us
chat with AGRIBOT
action_listen0.40
/agribot
C
agribot1.0

Phase-3

Problem identification and its significance

3.1 Problem identification and challenges

The share of **agriculture** in **GDP** being close to around 20 per cent in 2020-21, and also with the generations changing, the shortage of people practising agriculture exponentially declining ,economically speaking “less supply more demand” urged our strong gut to believe this is the future trending field maybe 5 from now or 10 years to be optimistic ,this all vital points triggered the team to work on the important phases of agriculture to improve the farmers produce, hence helping both the farmers as well as the nation to prosper.

After rigorous research and analysis, the main challenges that were identified to be of utmost importance for the farmers was to find solutions in the domains of selling their agricultural produce for reasonable prices, efficient fertilizer usage for crops and proper crops to be grown in their fields to get maximum produce.

But the task to be done is not as simple as it looks(**“Easier said than done”**), here are the major challenges to be looked upon:

- **Lack of proper and structured training data**: Now the problem here lies in the fact that most of the data collected from the government portals are raw data. Hence, now to train our model for the crop recommendation, we need to filter the important feature vectors to be used and apply the techniques such as mean normalization to make the ML model achieve a globally optimized solution.

- **Limited API access to government data**: The second challenge that was faced by the team was that only a limited datasets from the government portals were allowed access for public API calls. Also getting access to real world data,which is like an **important asset** these days is **expensive** since the world is on the verge of cutting-edge technologies.So, it is really hard to gather the data that we are badly in need of due to constraints like cost,privacy,credibility and other valuable concerns.

- **The very new RASA framework**: Although RASA is a powerful Machine Learning framework to build complex models, it was very recent and new(found in 2016), hence none of our team members knew about how to code and also about its features. Hence the team had to completely understand the technology from scratch. The team also struggled a lot to find solutions for errors due to very less resources available online as well as due to less community members available for the framework to discuss the issues.

- **Problem of deployment**: Any project becomes completed only when it reaches the public to experiment with it and also help the development team to improve the features of the chatbot. An important issue faced by the team at this stage was that most of the online resources and RASA community deployed the model on GCP, but we were unable to access it because of its hard policies on credit card(being students, really hard to possess credit cards since no source of income),this problem compelled the team to migrate to AWS,where the references or resources to deploy RASAX-SERVER was not even available on the official rasa documentation as well and on other side, scarcity of being charged heavily restricted our scope of experimentation.

3.2 Requirements and specifications

Hardware & OS Requirements:

Here are the minimum and recommended hardware specs and OS requirements:

	Install script	**Manual installation**
Operating System	Ubuntu 16.04 / 18.04 / 19.10 Debian 9 / 10 CentOS 7 / 8 RHEL 8	a modern Linux or Windows distribution that can run Docker
vCPUs	Minimum: 2 vCPUs Recommended: 2-6 vCPUs	

RAM	Minimum: 4 GB RAM Recommended: 8 GB RAM	
Disk Space	Recommended: 100 GB disk space available	

Port requirements:

Port	**Service**
22SSH	SSH access.
80HTTP	Web application access.
443HTTPS	Web application over HTTPS access

Supported Browsers:

The web interface aims to support browsers that meet the following criteria:

- 0.2% market share
- not Internet Explorer
- not Opera Mini

Software Requirements:

Operating System : Windows and linux

Technology : PYTHON, RASA

Dependencies: pickle, pandas, numpy, scikit-learn,matplotlib, seaborn

3.3 Installing RASA and RASAX

3.3.1 Installing RASA

Quick Installation

```
pip3 install -U pip
pip3 install rasa
```

You can create a new project by running: rasa init

Step-by-step Installation Guide

You can follow official rasa documentation part (link to be included)

3.3.2 Installing RASA X

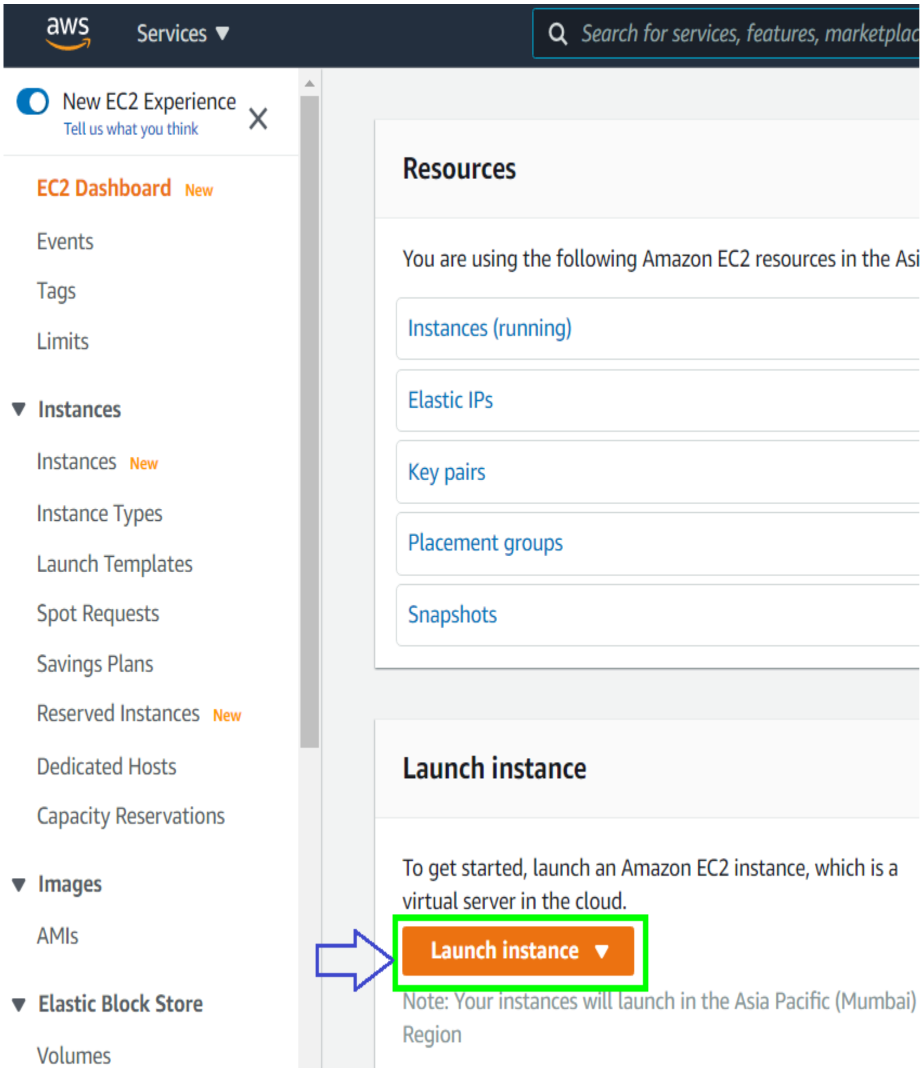
Rasa X:

- layers on top of Rasa Open Source and helps you build a better assistant
- is a free, closed source tool available to all developers
- can be deployed anywhere, so your training data stays secure and proprietary

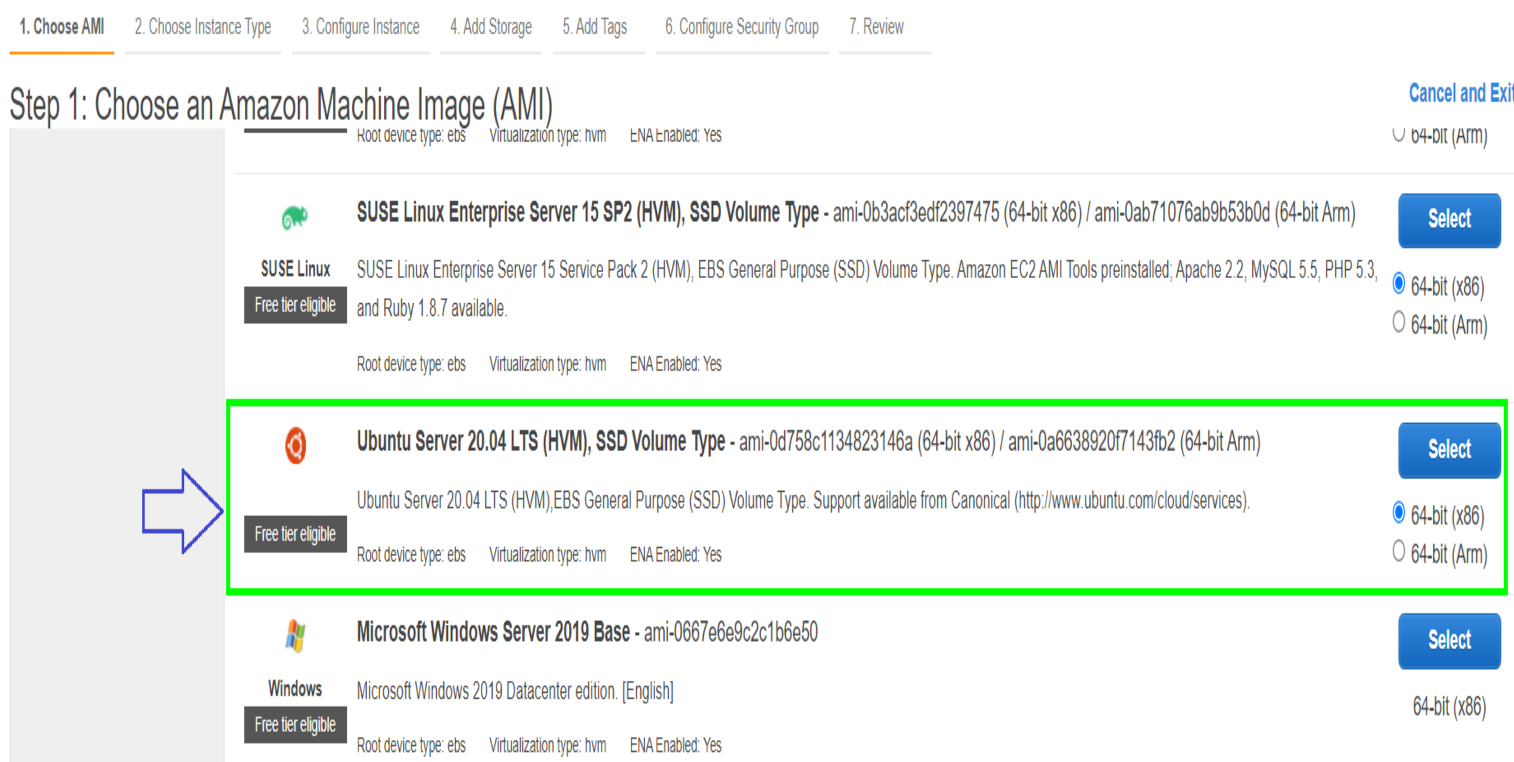
3.3.3 Deploying Rasa X

Configure the VM instance

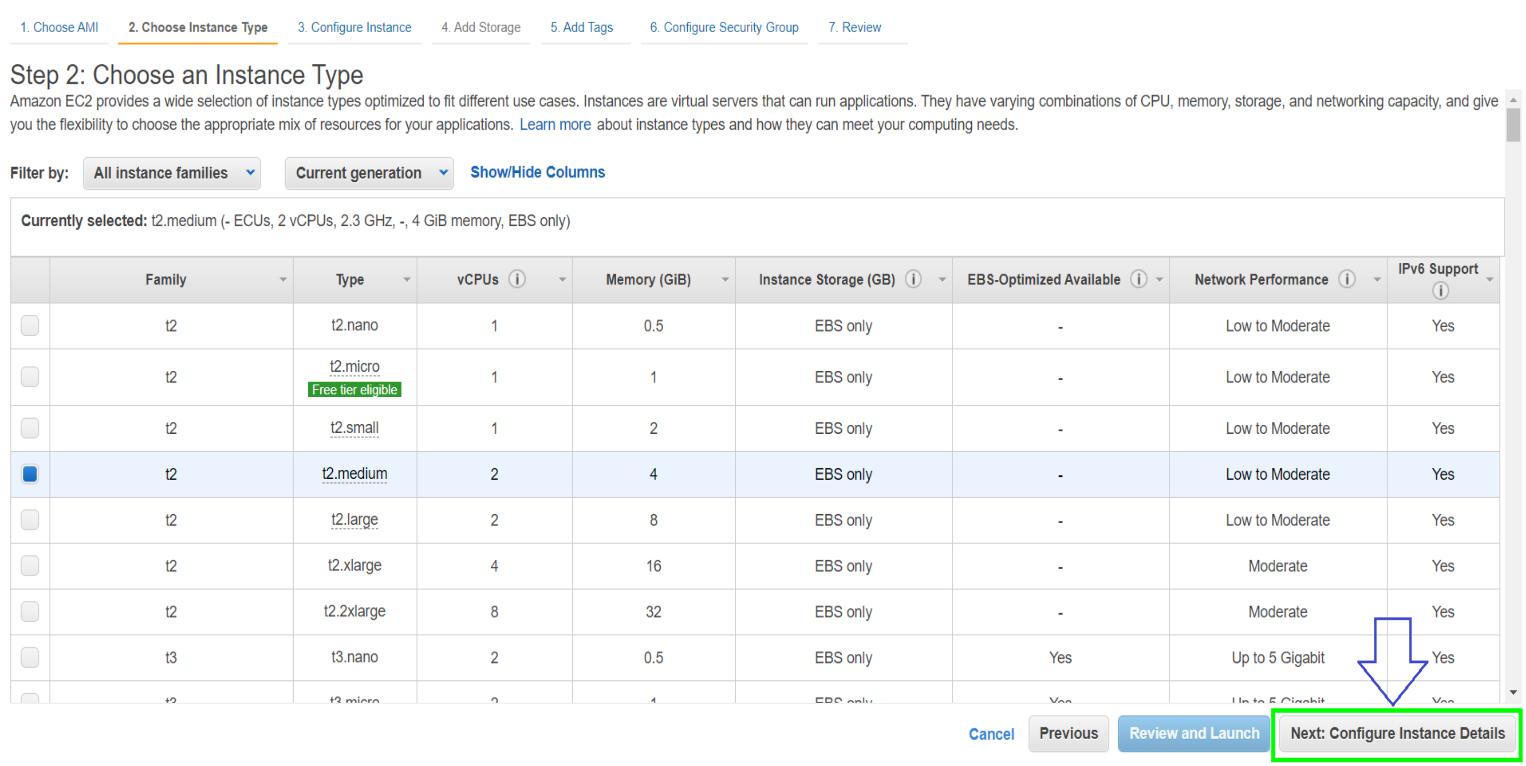
Step 1 : Log in to your AWS Console and navigate to Services-> Compute-> EC2. Click Launch Instance.



Step 2 : Choose an Amazon Machine Image(AMI)->Go to Ubuntu Server 20.04 LTS>select.

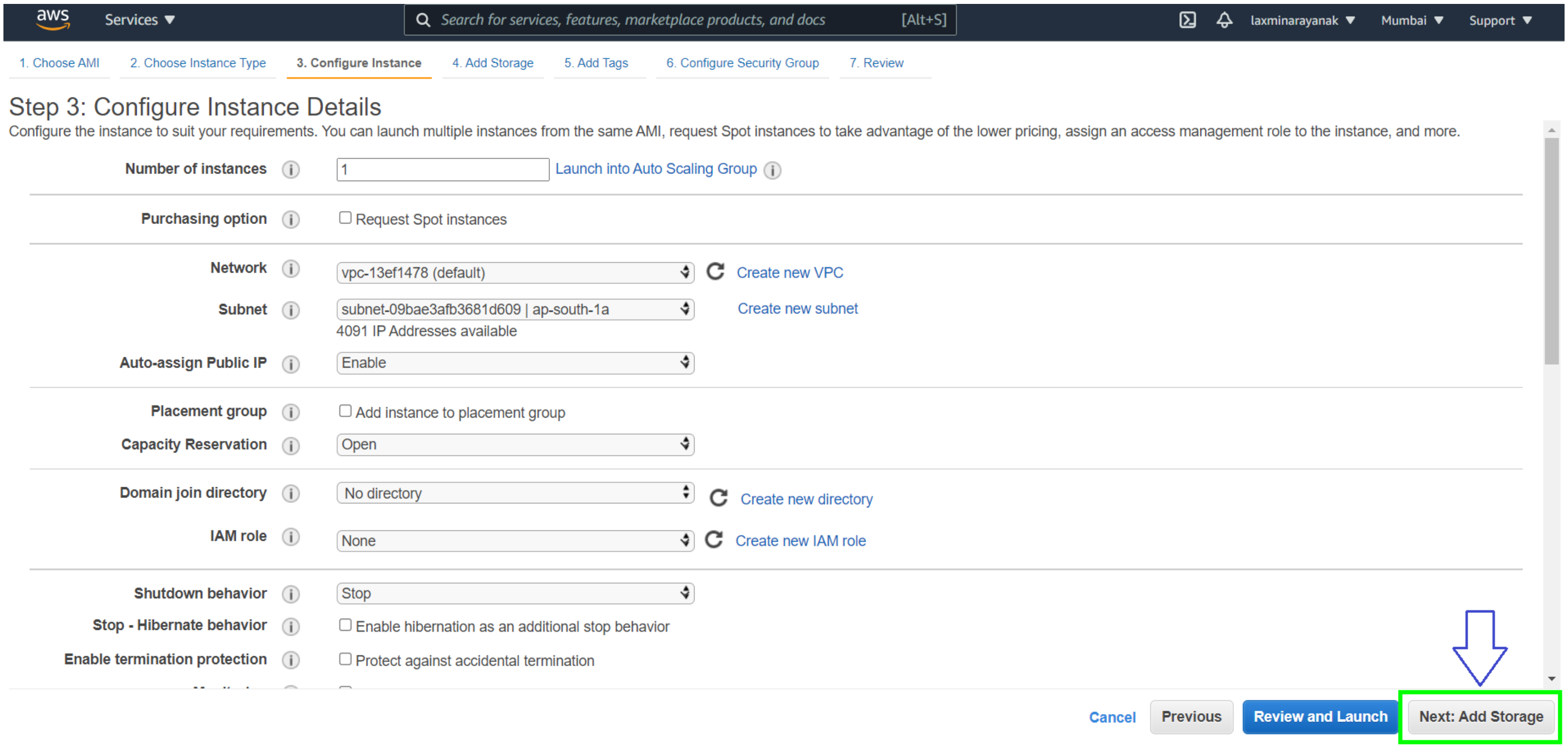


Step 3 : Choose an instance type as **t2.medium** ->Configure Instance Details

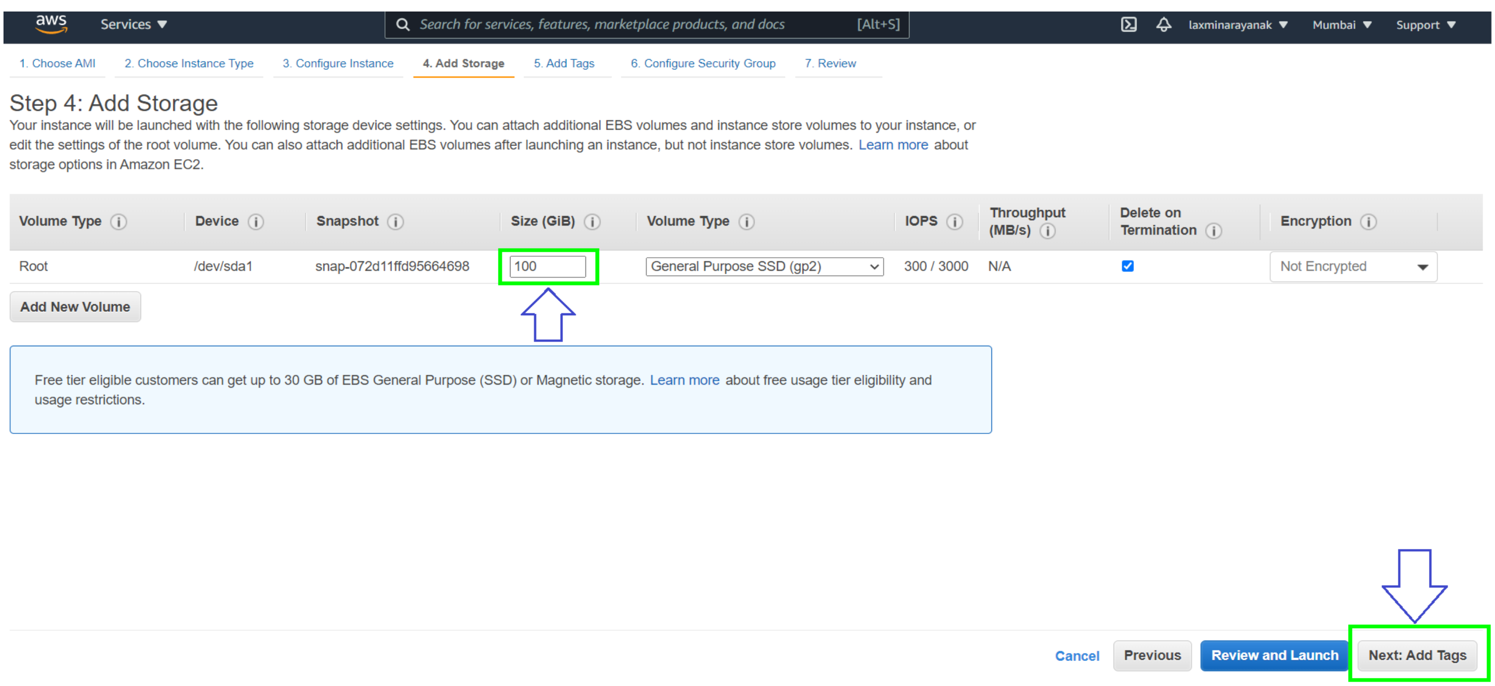


Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
t2	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
t2	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate	Yes
t2	t2.small	1	2	EBS only	-	Low to Moderate	Yes
t2	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
t2	t2.large	2	8	EBS only	-	Low to Moderate	Yes
t2	t2.xlarge	4	16	EBS only	-	Moderate	Yes
t2	t2.2xlarge	8	32	EBS only	-	Moderate	Yes
t3	t3.nano	2	0.5	EBS only	Yes	Up to 5 Gigabit	Yes

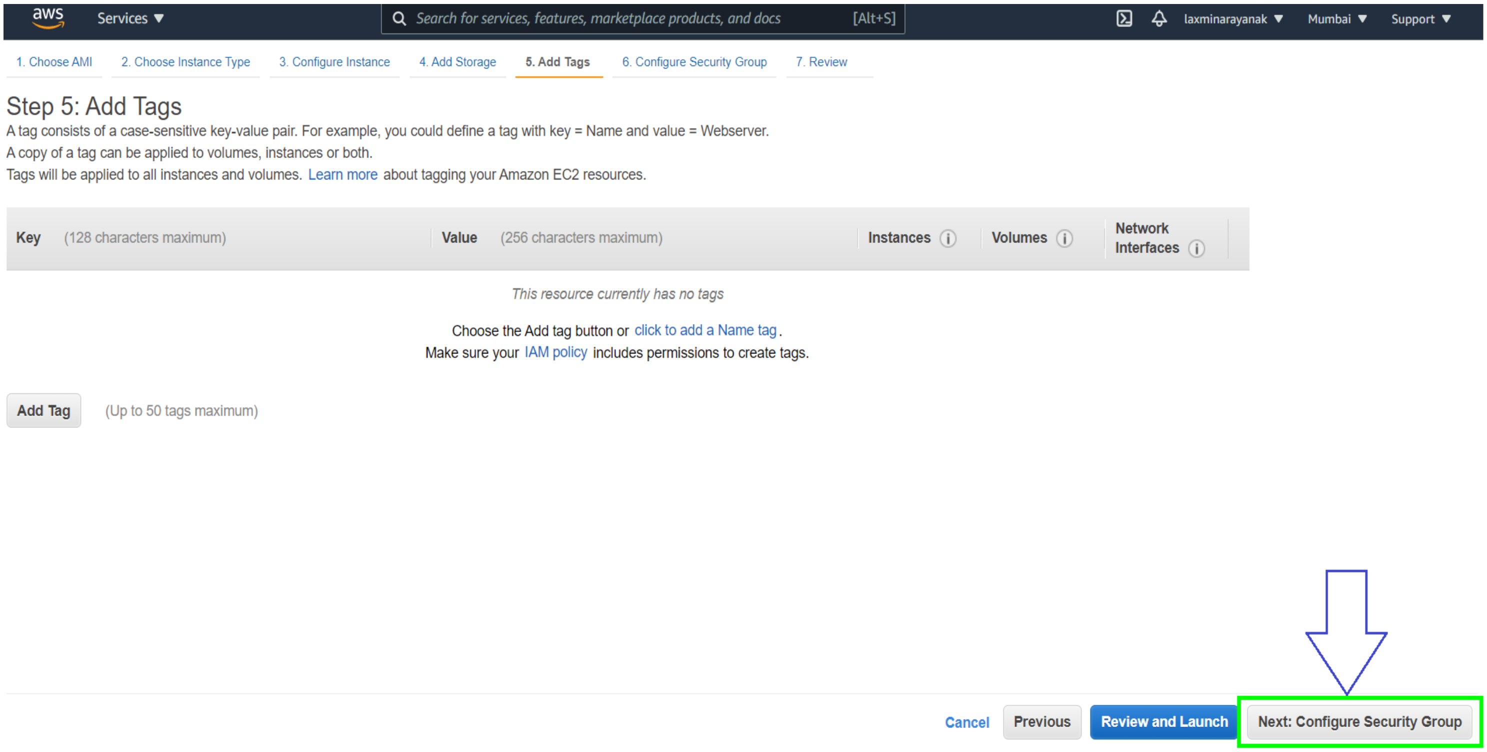
Step 4 : Keep all default settings as it is and click on **Next:Add Storage**



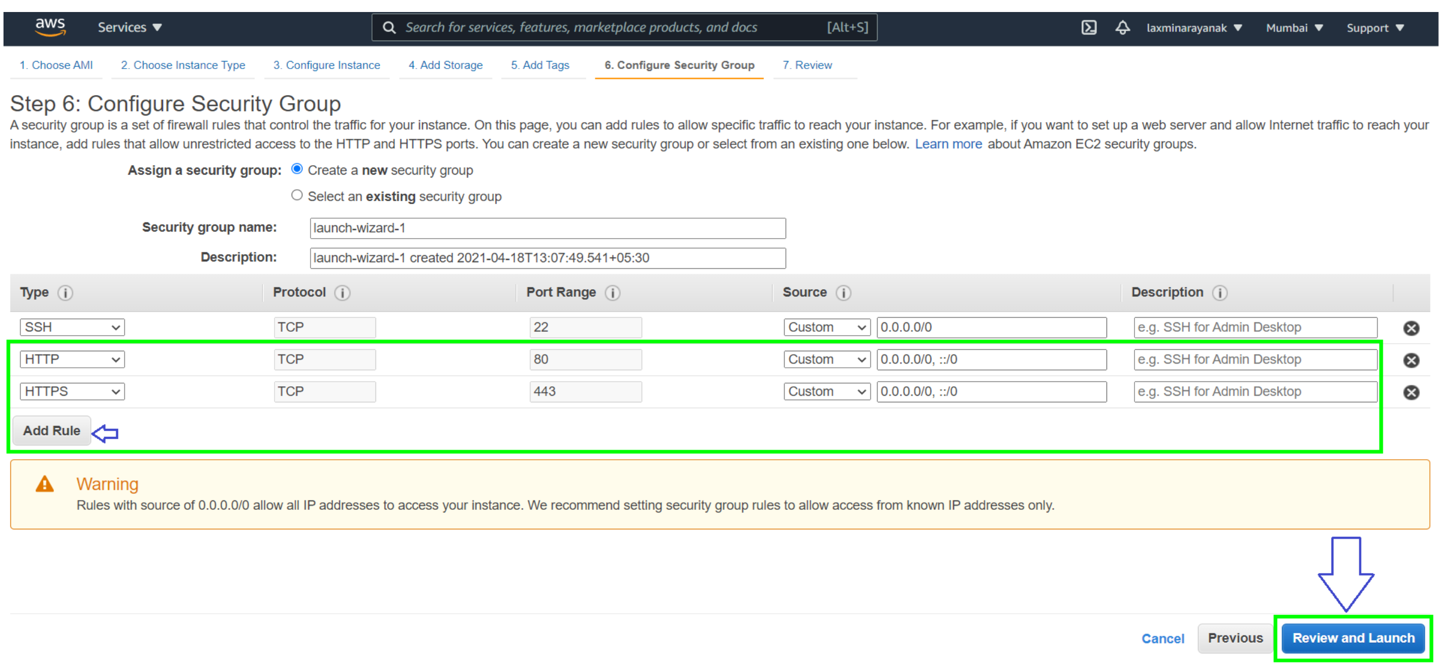
Step 5 : Here change **size(GiB)** to 100 and then click on **Next:Add Tags**



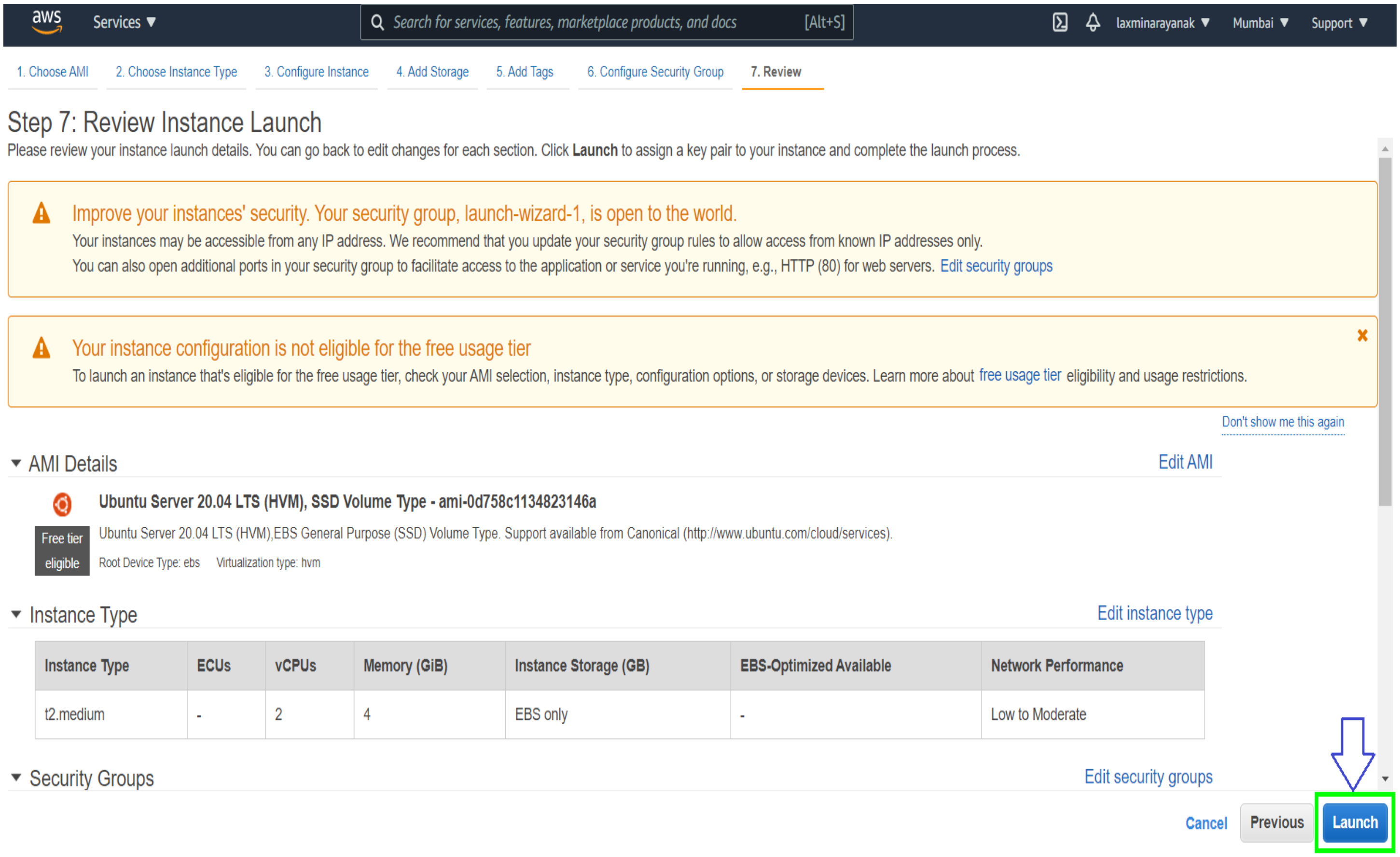
Step 6 : No changes here just click on **Next:Configure Security Group**

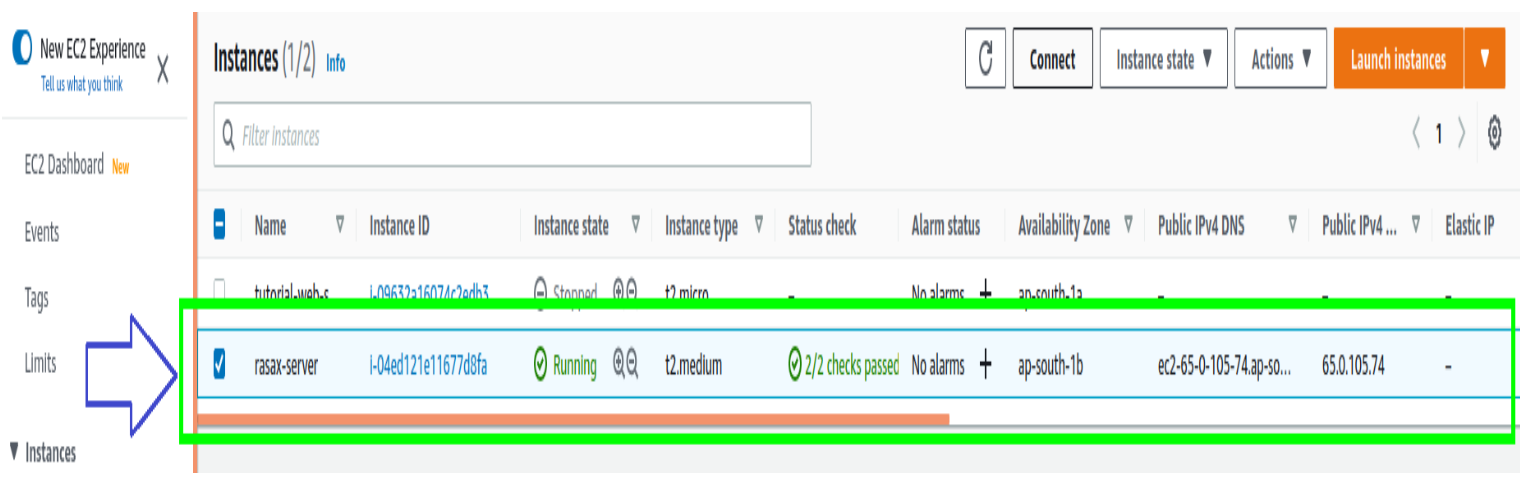


Step 7 : Here Add two Rules **HTTP** and **HTTPS** and then click on **Review and Launch**

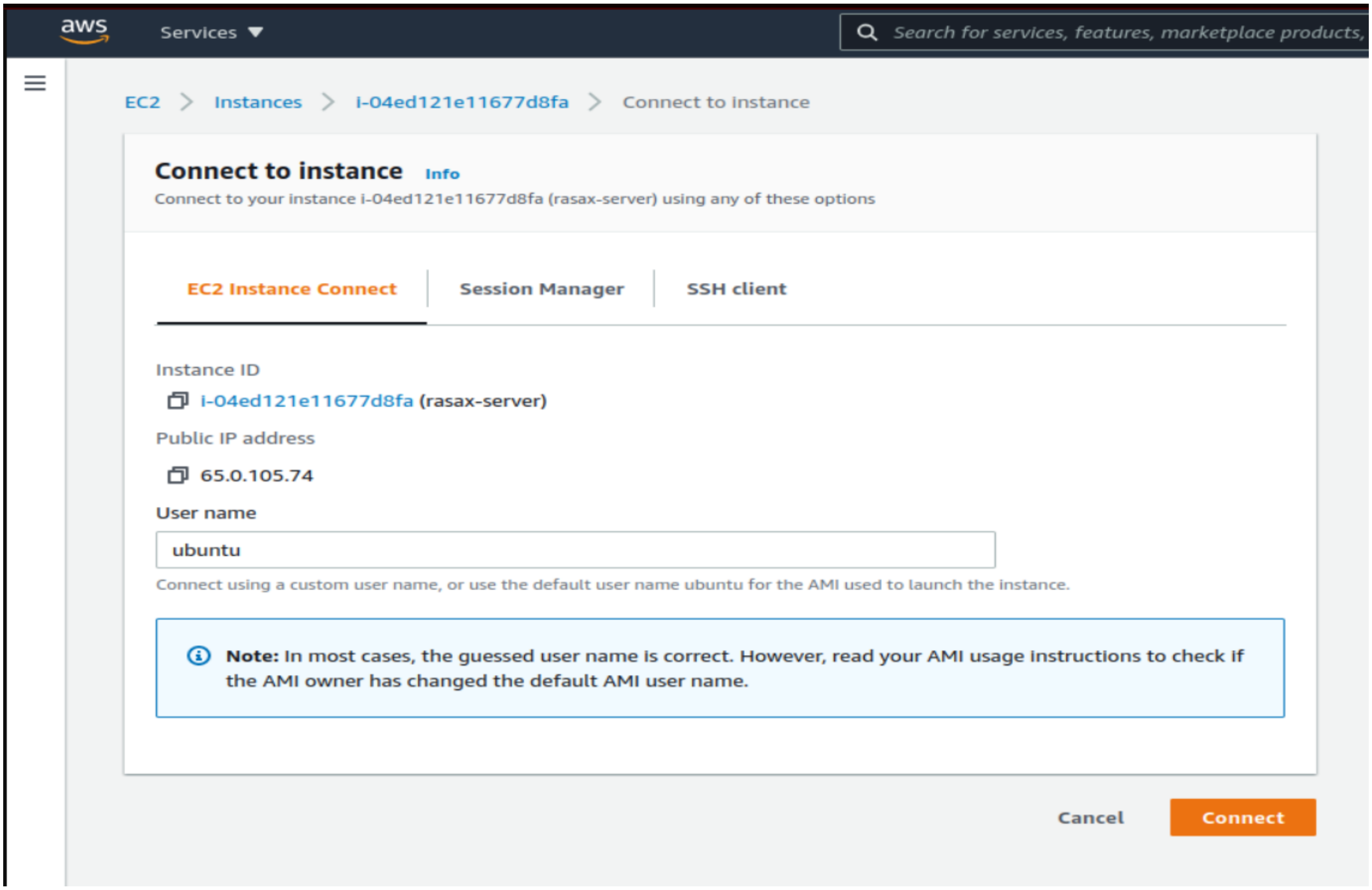


Step 8 : Just click on **Launch** and your instance will be created.

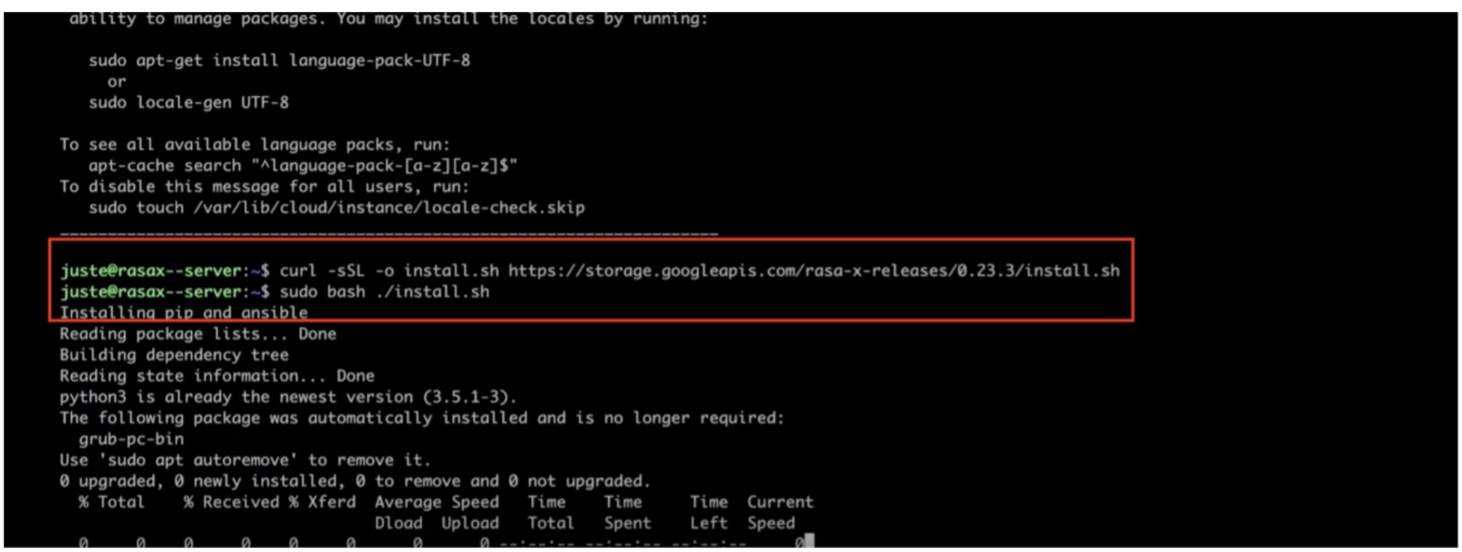




Step 9 : Connect to your created instance

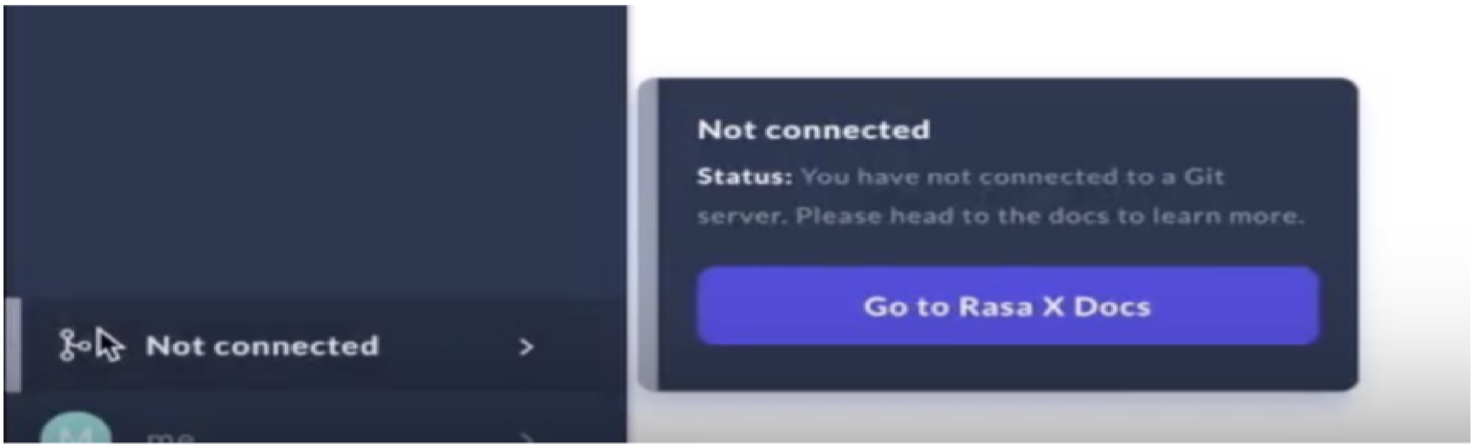


Step 10: Install the required dependencies

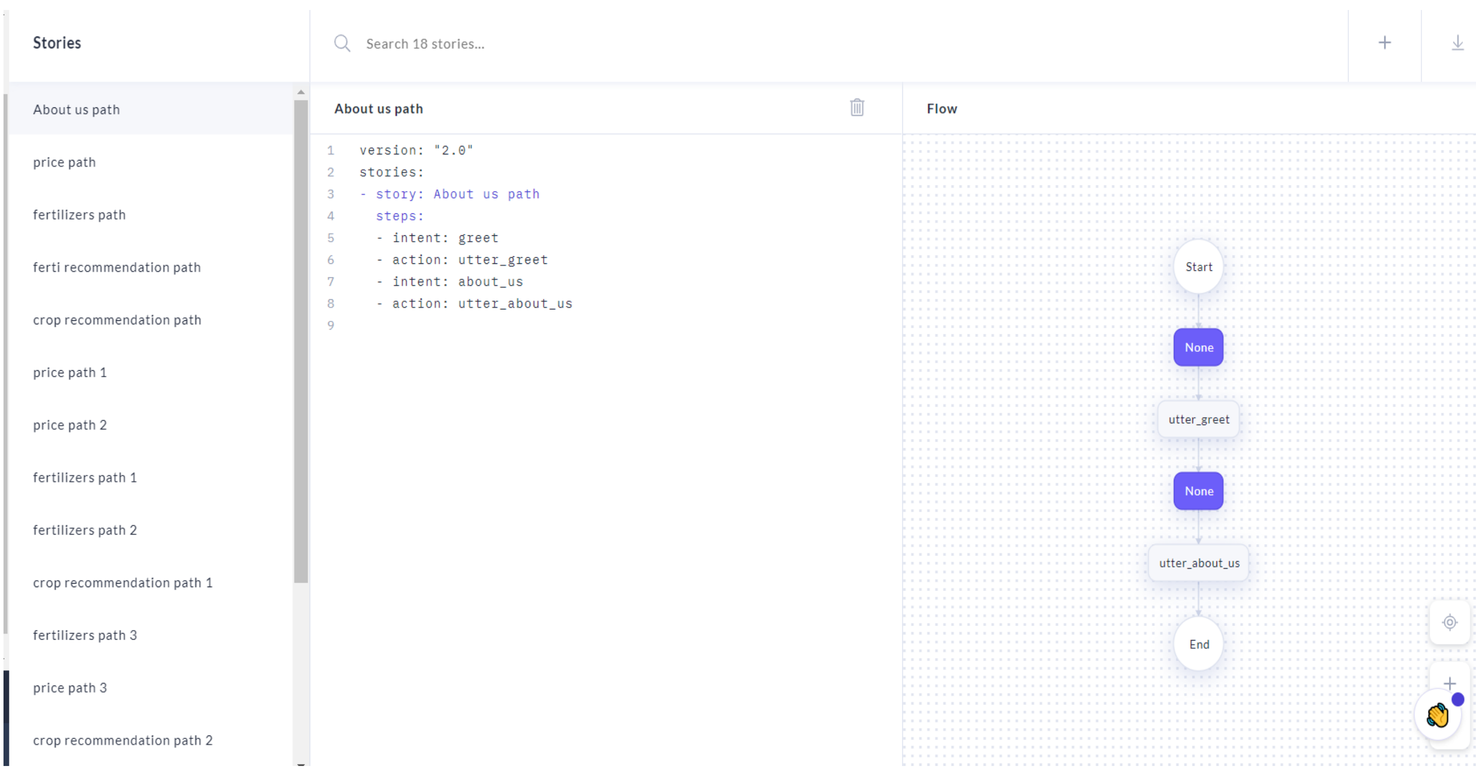


Step 11: Check for the files present in the rasa folder.

Step 12: Open the ipv4 address of the created instance and connect to the github repository.



Step 13: Eureka! Now all the features of rasa x can be found and model can be improved by sharing to multiple users.

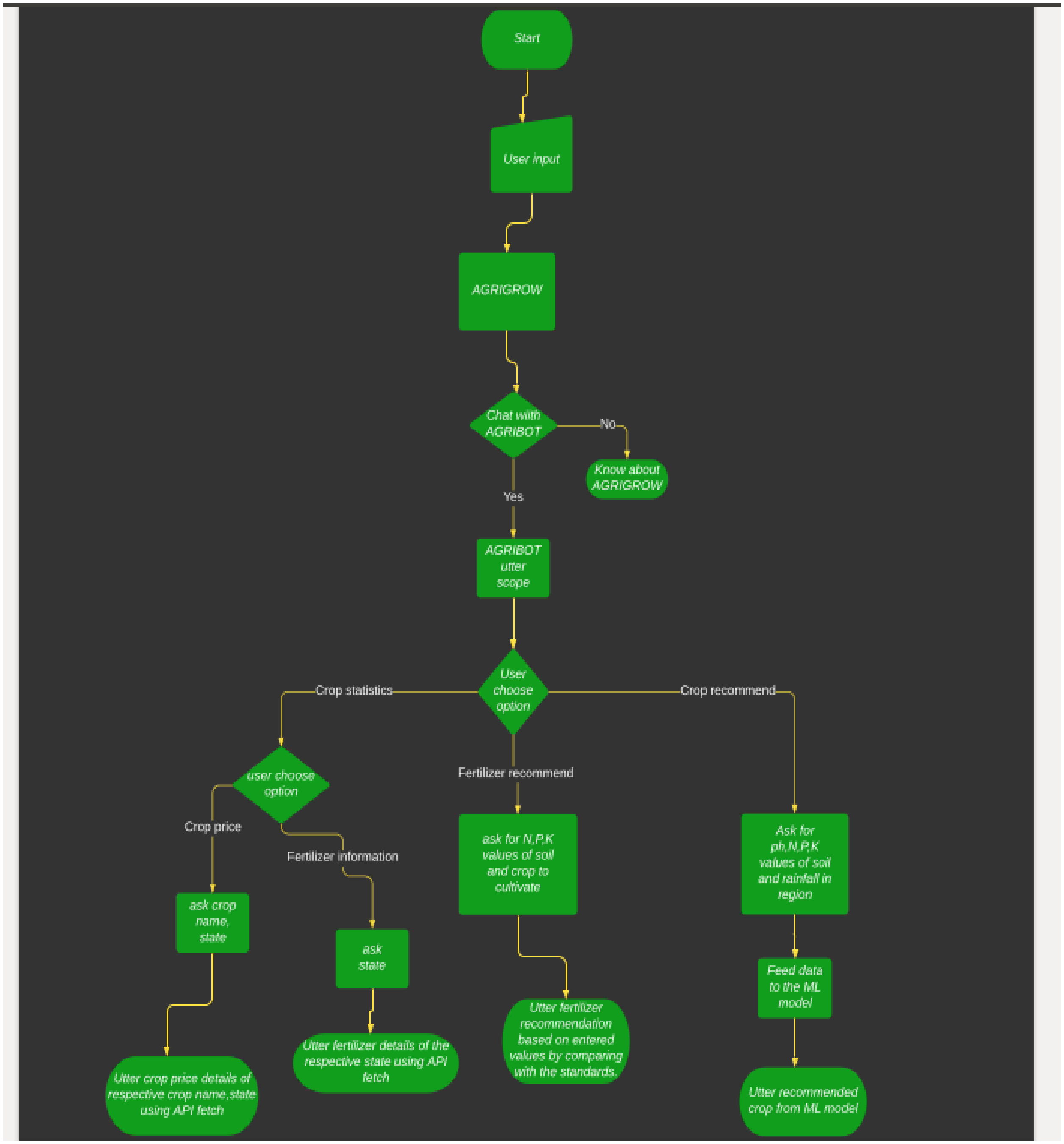


Phase-4

Proposed solution and implementation

4.1 Approach to solve problems in the specified domains

Project Flow chart:



As mentioned in the problem description, the main areas of focus is on the fertilizer recommendation, crop recommendation, etc. The important queries required by the farmers were taken into account and around 5 story paths were designed accordingly. Let's have a look at each of the story paths.

1. **About us path:**

```
- story: About us path
  steps:
  - intent: greet
  - action: utter_greet
  - intent: about_us
  - action: utter_about_us
```

2. **Crop price path(implemented using api calls):**
    - This is one of the paths that uses an api call. We have taken 3 major crops(rice, cotton and jute) and use an api call to the data.gov.in website.
    - example(https://api.data.gov.in/resource/6e8e9a24-491d-4bcb-bdf4-bb0724cbb926?api-key=579b464db66ec23bdd000001cdd3946e44ce4aad7209ff7b23ac571b&format=json&offset=0&limit=100&filters[state_ut]=Andhra Pradesh)where api-key is the unique key obtained by signing into the data.gov.in website to gain access to read information from the govt database.
    - But instead of Andhra Pradesh we provide the state that was extracted from the farmer during conversations in rasa. This gives us the output in the form of a json format.
    - Now we can extract only the required information that is needed by using indexingjustlikearrays.(ex:current['records'][0]['_2017_18___prod__as_per_cab_meeting_dt__18_6_19__qty__in_lakh_bales_'] extracts from the govt database only the produce as per cab meeting in lakh bales in the respective state).
    - Once this step is over we can display only the important information extracted about the crop and the state in which it was grown as responses uttered by the chatbot.

```
- story: price path
  steps:
  - intent: greet
  - action: utter_greet
  - intent: agribot
  - action: utter_agribot
  - intent: statistics
  - action: utter_statistics
  - intent: price
  - action: utter_ask_crop1
  - intent: crop_entry
  - action: utter_tc_as
  - intent: state_entry
  - action: action_get_price
```

3. Fertilizer path:

- We follow the same steps as mentioned in price path to extract information about fertilizers as well using api calls.
- https://api.data.gov.in/resource/1a800a9a-7c6e-42ba-b238-6ae1c17d5195?api-key={}&format=json&offset=0&limit=10&filters[state_u_t_]={}.format(api_key,loc) this is

```
- story: fertilizers path
  steps:
  - intent: greet
  - action: utter_greet
  - intent: agribot
  - action: utter_agribot
  - intent: statistics
  - action: utter_statistics
  - intent: ferti
  - action: utter_ferti
  - intent: state_entry
  - action: action_get_fertilizers
```

4. **Fertilizer recommendation path:**

- This is a path which uses the recommendations that was provided by a crop health expertise website.
- We have stored the details collected in the form of a dictionary and call the specific information in the dictionary based on the level of nutrients the soil(N,P,K values) has when compared to the national standard mentioned in the website and provide the specified suggestions based on that.

```
- story: ferti recommendation path
  steps:
  - intent: greet
  - action: utter_greet
  - intent: agribot
  - action: utter_agribot
  - intent: ferti_recommendation
  - action: utter_ask_phos
  - intent: phosphorous_entry
    entities:
    - phosphorous: P_30
  - slot_was_set:
    - phosphorous: P_30
  - action: utter_ask_crop
  - intent: crop_entry
  - action: utter_ask_nitrogen
  - intent: nitrogen_entry
    entities:
    - nitrogen: N_20
  - slot_was_set:
    - nitrogen: N_20
  - action: utter_ask_potas
  - intent: potassium_entry
    entities:
    - potassium: K_40
  - slot_was_set:
    - potassium: K_40
  - action: action_ferti_recommendation
```

5. Crop recommendation path:

In the crop recommendation application, the user can provide the soil data from their side and the application will predict which crop should the user grow.

- This is the most essential part of the AI chatbot which gives suggestions about what crop to grow using predictions by an ML model. We shall see a detailed analysis of the various ML models that were used and their respective scores in the coming section.

```
- story: crop recommendation path
  steps:
  - intent: greet
  - action: utter_greet
  - intent: agribot
  - action: utter_agribot
  - intent: recommend
  - action: utter_ask_ph
  - intent: power_hydrogen
    entities:
    - ph: ph_5.65
  - slot_was_set:
    - ph: ph_5.65
  - action: utter_ask_nitrogen
  - intent: nitrogen_entry
    entities:
    - nitrogen: N_20
  - slot_was_set:
    - nitrogen: N_20
  - action: utter_ask_phos
  - intent: phosphorous_entry
    entities:
    - phosphorous: P_30
  - slot_was_set:
    - phosphorous: P_30
  - action: utter_ask_city
  - intent: cities
    entities:
    - city: chennai
  - slot_was_set:
    - city: chennai
  - action: utter_ask_potas
  - intent: potassium_entry
    entities:
    - potassium: K_40
  - slot_was_set:
    - potassium: K_40
  - action: utter_ask_rainfall
  - intent: rainfall_entry
    entities:
    - rainfall: rf_263.72
  - slot_was_set:
    - rainfall: rf_263.72
  - action: action_crop_recommendation
```

4.2 The ML models and their scores

4.2.1 Data extraction and normalization:

This step is needed because the data extracted from govt portals being completely raw, we need to extract only the required features to train our model. Now from the fertilizers.csv(N,P,K,ph as features) and cropdata.csv(temperature,humidity,ph and rainfall) areto be merged so that more features can be used to train our data model. The labels of the final model are as follows:

array(['rice', 'wheat', 'mungbean', 'tea', 'millet', 'maize', 'lentil',
'jute', 'coffee', 'cotton', 'groundnut', 'peas', 'rubber',
'sugarcane', 'tobacco', 'kidneybeans', 'mothbeans', 'coconut',
'blackgram', 'adzukibeans', 'pigeonpeas', 'chickpea', 'banana',
'grapes', 'apple', 'mango', 'muskmelon', 'orange', 'papaya',
'pomegranate', 'watermelon'], dtype=object)

4.2.2 The Crop recommendation model:

The following are the features that are used by our ML model for crop recommendation:

Index(['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall', 'label']

Comparing accuracy from different ML models that were built:

1. **Gaussian naive bayes:**

Guassian Naive Bayes

54]:
```
from sklearn.naive_bayes import GaussianNB

NaiveBayes = GaussianNB()

NaiveBayes.fit(Xtrain,Ytrain)

predicted_values = NaiveBayes.predict(Xtest)
x = metrics.accuracy_score(Ytest, predicted_values)
acc.append(x)
model.append('Naive Bayes')
print("Naive Bayes's Accuracy is: ", x)

print(classification_report(Ytest,predicted_values))
```

```
Naive Bayes's Accuracy is:  0.990909090909091
```

2. Decision tree:

Seperating features and target label

In [46]:
```
features = df[['N', 'P','K','temperature', 'humidity', 'ph', 'rainfall']]
target = df['label']
#features = df[['temperature', 'humidity', 'ph', 'rainfall']]
labels = df['label']
```

In [47]:
```
# Initialzing empty lists to append all model's name and corresponding name
acc = []
model = []
```

In [48]:
```
# Splitting into train and test data

from sklearn.model_selection import train_test_split
Xtrain, Xtest, Ytrain, Ytest = train_test_split(features,target,test_size = 0.2,random_state =2)
```

Decision Tree

In [49]:
```
from sklearn.tree import DecisionTreeClassifier

DecisionTree = DecisionTreeClassifier(criterion="entropy",random_state=2,max_depth=5)

DecisionTree.fit(Xtrain,Ytrain)

predicted_values = DecisionTree.predict(Xtest)
x = metrics.accuracy_score(Ytest, predicted_values)
acc.append(x)
model.append('Decision Tree')
print("DecisionTrees's Accuracy is: ", x*100)

print(classification_report(Ytest,predicted_values))
```

```
DecisionTrees's Accuracy is:  90.0
```

3. Support Vector Machine:

Support Vector Machine (SVM)

57]:
```
from sklearn.svm import SVC

SVM = SVC(gamma='auto')

SVM.fit(Xtrain,Ytrain)

predicted_values = SVM.predict(Xtest)

x = metrics.accuracy_score(Ytest, predicted_values)
acc.append(x)
model.append('SVM')
print("SVM's Accuracy is: ", x)

print(classification_report(Ytest,predicted_values))
```

```
SVM's Accuracy is:  0.10681818181818181
```

4. **Logistic Regression**

Logistic Regression

```
from sklearn.linear_model import LogisticRegression

LogReg = LogisticRegression(random_state=2)

LogReg.fit(Xtrain,Ytrain)

predicted_values = LogReg.predict(Xtest)

x = metrics.accuracy_score(Ytest, predicted_values)
acc.append(x)
model.append('Logistic Regression')
print("Logistic Regression's Accuracy is: ", x)

print(classification_report(Ytest,predicted_values))
```

```
Logistic Regression's Accuracy is:  0.9522727272727273
```

5. **Random Forest:**

Random Forest

```
from sklearn.ensemble import RandomForestClassifier

RF = RandomForestClassifier(n_estimators=20, random_state=0)
RF.fit(Xtrain,Ytrain)

predicted_values = RF.predict(Xtest)

x = metrics.accuracy_score(Ytest, predicted_values)
acc.append(x)
model.append('RF')
print("RF's Accuracy is: ", x)

print(classification_report(Ytest,predicted_values))
```

```
RF's Accuracy is:  0.990909090909091
```

6. XGBoost:

XGBoost

```
import xgboost as xgb
XB = xgb.XGBClassifier()
XB.fit(Xtrain,Ytrain)

predicted_values = XB.predict(Xtest)

x = metrics.accuracy_score(Ytest, predicted_values)
acc.append(x)
model.append('XGBoost')
print("XGBoost's Accuracy is: ", x)

print(classification_report(Ytest,predicted_values))
```

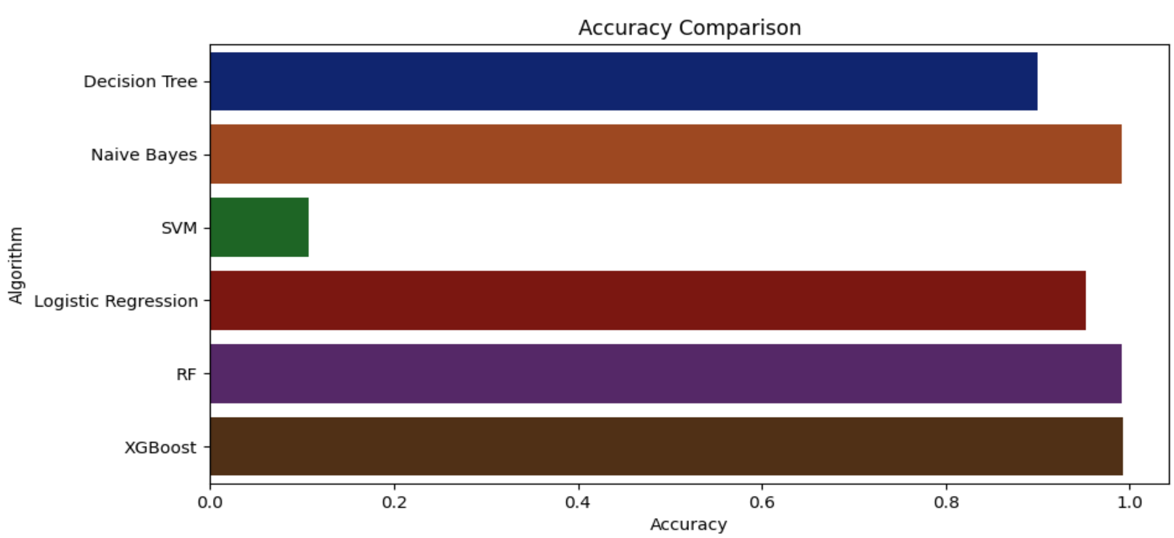
```
XGBoost's Accuracy is:  0.9931818181818182
```

Final accuracy comparison of all the models:

Accuracy Comparison

```
plt.figure(figsize=[10,5],dpi = 100)
plt.title('Accuracy Comparison')
plt.xlabel('Accuracy')
plt.ylabel('Algorithm')
sns.barplot(x = acc,y = model,palette='dark')
```

```
<AxesSubplot:title={'center':'Accuracy Comparison'}, xlabel='Accuracy', ylabel='Algorithm'>
```



As from the above figure, we can say that random provides a significant accuracy level for our ML model to make proper predictions, hence we save this model into a pickle file and import that into rasa to help our bot display the predicted crop.

Phase-5

CONCLUSION AND FUTURE SCOPE

5.1 Conclusions:

We have used the AI chatbot to its newest level into the field where it was least exposed(Agricultural sector, Farming). We hope to have created a foundation for many similar future developments to help the Agricultural sector progress and prosper with such new and fascinating technologies. This project has also shown that AI chatbots can not only be used for business management, but can also revolutionize and change the way that the current Agricultural system works.

This project was a genuine attempt by the team members to make AI chatbots revolutionize the present agricultural system with utmost diligence.

5.2 Future scope:

1. The language used to implement the project was in english, this can however be extended to multiple languages with an option for the farmer to choose from with the chatbot recommendations also in the language chosen by the farmer.

2. With more amount of training data in hand, the model could be trained rigorously and also with more input features such as the season in which the crop is grown, etc.

3. A voice can also be added to the utterances by the bot in the native language which the farmer chooses(such as Alexa,Siri,etc.)

4. Another feature of enabling the farmers to sell their produce in the nearest available shop that provides the best rates and also connects the farmers to the best fertilizer selling shops in their locality. (This feature could involve integrating our chatbot with google maps).

5.3 How to use our repository and contribute to our project

Step 1: Clone our github repository into your desired location using the below link: https://github.com/kuluruvineeth/Agrosahakar

Step 2: create a VM instance on any cloud platform such as aws, azure, GCP, heroku.

Step 3: Deploy RASAX server on VM instance created in step1 as mentioned in the phase 2. For detailed information please refer to the official RASA documentation:RASA documentation

Step 4: Now your RASAX server will be up and running.

Step 5: Connect your repository to the RASAX server:

1. **Generate SSH keys:**
   - Navigate back to your terminal. If you've closed the connection to your VM instance, log back in.
   - Run the following command to generate a public and private SSH key

```
ssh-keygen -t rsa -b 4096 -f git-deploy-key
```

   - After the key has finished generating, you can run the ls command in the `/rasa/etc` directory to see the newly created keys: git-deploy-key (the private key) and git-deploy-key.pub (the public key).

2. **Save the public key in GitHub:**
   - We'll print the public key to the terminal so we can copy and save it in our GitHub settings.Run the following command to view the public key:

```
cat git-deploy-key.pub
```

   - Copy the entire contents.

In your GitHub repository, navigate to Settings>Deploy keys. Click the Add deploy key button and paste your public key into the Key box. Give the key a title to identify it, like medicare-rasax, and be sure to check the box to allow Write permissions. Click Add key.

3. We'll establish the connection between the Rasa X instance and GitHub repository by making a POST request to this Rasa X API endpoint.
4. The JSON request body contains three pieces of information:
   - **repository_url** - The SSH URL for your GitHub repository, e.g. `kuluruvineeth/Agrosahakar.git`
   - To get the URL for your repo, click the Clone or download button on your GitHub repository and select the Use SSH link.
   - **target_branch** - The GitHub repository branch where Rasa X should push and pull changes, e.g. master
   - **ssh_key** - The private SSH key generated on your server.

   To copy the private key, run the following command in the `/etc/rasa` folder on your server:

```
cat git-deploy-key
```

Copy the entire contents of the key, including the lines

`-----BEGIN RSA PRIVATE KEY-----`

And

`-----END RSA PRIVATE KEY-----`

Once you've assembled the JSON object, you'll have something like this:

```
{ "repository_url":"kuluruvineeth/Agrosahakar.git", "target_branch": "master",
"ssh_key": "-----BEGIN RSA PRIVATE
KEY-----b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAA
AABAAACFwAAAAdzc2gtcnNhAAAAAwEAAQAAAgEAu/Giin7t8DFMxsaTb
yy1To2EQpLIAhpAIgpyC/e45NYVTwKRGCB1mxHzt5IWoh7GSWry3pKFBM7
```

```
4UpXxrRPBdCmFeUIiJoslAukNkRSckAUj0VEfOIZLf2SSPg...CDHniFksE1Sjk
AAAEBANJacZeM2Qdk/vditmBQV97Ac2VJL/Btt8Rks2Vb3CORyXQn3Bpb+5
ZONhmPEoCg4FcZbAm02gYw3dSoBBWz2i8mmAv71mVsNoddWKpDngRFv4
PUaITnYYxrZ4-----END RSA PRIVATE KEY-----"}
```


We'll save this JSON object in a file called repository.json, in the `/rasa/etc` folder on the server

5. First, let's create that file `touch repository.json`
   - Open the file to edit it:`nano repository.json`

   Paste the JSON object into the file. Press Control + X to exit the editor, and confirm Y to save your changes when prompted.

   - Head back to the terminal. Still in the `/etc/rasa` directory, run the following cURL command which you will get clicking on upload model button in RASAX interface, replacing the Rasa X server URL and API key values with your own:

```
curl --request POST \ --url http://<Rasa X server
host>/api/projects/default/git_repositories?api_token=<your api token> \ --header
'content-type: application/json' \ --data-binary @repository.json
```

   - Check the connection by navigating back to the Rasa X dashboard in your browser and checking the Integrated Version Control icon in the bottom left corner. If the connection was successful, you'll see either a green indicator, meaning Rasa X is up to date with the GitHub repository, or a yellow indicator, meaning Rasa X has changes that need to be pushed to GitHub.

Step 6: Set up the Actions Server:

- We have one more thing to configure: the assistant's custom action server. To do this, we'll place the assistant's custom action code within an `actions` directory on the server.
- Connect to your server and make sure you're in the /etc/rasa directory. In your terminal, run the following commands to create the actions directory and two files inside it: `__init__.py` and `actions.py`:

- Run `nano actions/actions.py` to edit the newly-created actions.py file.
- Paste the code from your assistant's actions.py file into the blank file, save, and close the editor.
- Then, we need to create a docker-compose.override.yml file. This file instructs docker-compose to spin up a custom action server when the Rasa X server starts up.
- Let's create that file:

```
touch docker-compose.override.yml
```

Open the file editor:`nano docker-compose.override.yml`

And add the following contents:

```
version: '3.4'
services:
app:
image: 'rasa/rasa-sdk:latest'
volumes: - './actions:/app/actions' expose: - '5055'
depends_on: - rasa-production
```

- Here, we're using the rasa-sdk image to run our custom actions, and we're specifying that the actions server will listen on port 5055. The actions server depends on the rasa-production service, which is responsible for running the trained model, parsing intent messages, and predicting actions.

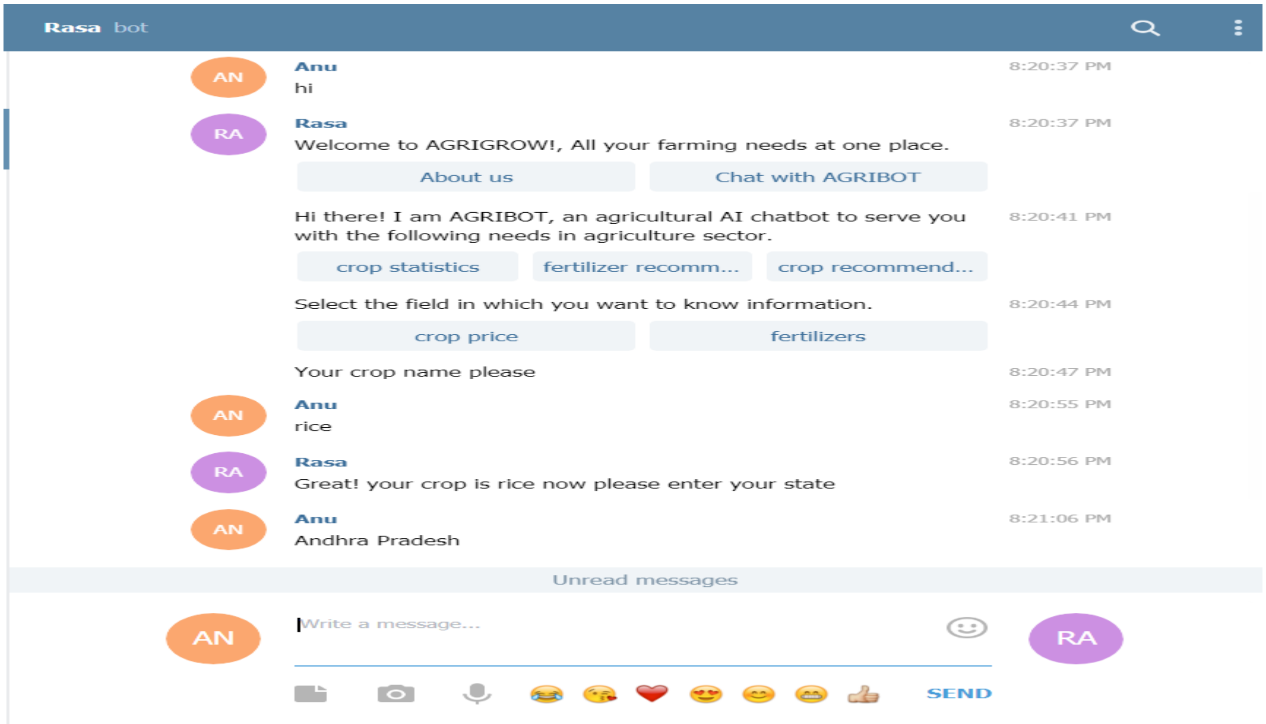
- Once you've saved the file, you can restart the Rasa X docker container and the assistant will be fully functional on Rasa X.

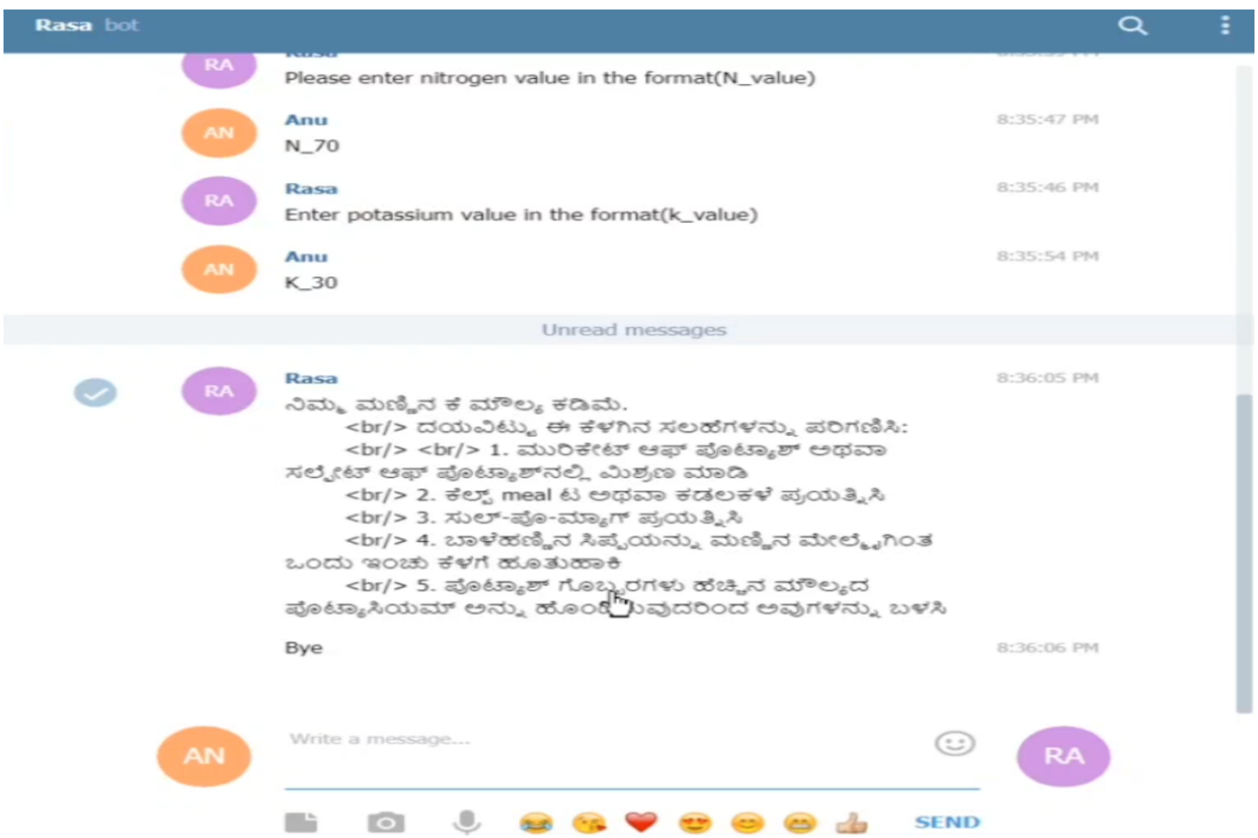
```
sudo docker-compose up -d
```

Step 7: Eureka!!! You have done the complete set up and are ready to use our chatbot. Feel free to share your comments in github.

More Screenshots

	Crop	N	P	K	pH	soil_moisture
0	rice	80	40	40	5.5	30
3	maize	80	40	20	5.5	50
5	chickpea	40	60	80	5.5	60
12	kidneybeans	20	60	20	5.5	45
13	pigeonpeas	20	60	20	5.5	45
14	mothbeans	20	40	20	5.5	30
15	mungbean	20	40	20	5.5	80
18	blackgram	40	60	20	5	60
24	lentil	20	60	20	5.5	90
60	pomegranate	20	10	40	5.5	30
61	banana	100	75	50	6.5	40
62	mango	20	20	30	5	15
63	grapes	20	125	200	4	60
66	watermelon	100	10	50	5.5	70
67	muskmelon	100	10	50	5.5	30
69	apple	20	125	200	6.5	50
74	orange	20	10	10	4	60
75	papaya	50	50	50	6	20
88	coconut	20	10	30	5	45
93	cotton	120	40	20	5.5	70
94	jute	80	40	40	5.5	20
95	coffee	100	20	30	5.5	20


Rasa bot
AN
Anu
hi
8:20:37 PM
RA
Rasa
Welcome to AGRIGROW!, All your farming needs at one place.
8:20:37 PM
About us
Chat with AGRIBOT
Hi there! I am AGRIBOT, an agricultural AI chatbot to serve you with the following needs in agriculture sector.
8:20:41 PM
crop statistics
fertilizer recomm...
crop recommend...
Select the field in which you want to know information.
8:20:44 PM
crop price
fertilizers
Your crop name please
8:20:47 PM
AN
Anu
rice
8:20:55 PM
RA
Rasa
Great! your crop is rice now please enter your state
8:20:56 PM
AN
Anu
Andhra Pradesh
8:21:06 PM
Unread messages
AN
Write a message...
RA
SEND


Rasa bot
RA
Please enter nitrogen value in the format(N_value)
AN
Anu
N_70
8:35:47 PM
RA
Rasa
Enter potassium value in the format(k_value)
8:35:46 PM
AN
Anu
K_30
8:35:54 PM
Unread messages
RA
Rasa
ನಿಮ್ಮ ಮಣ್ಣಿನ ಕೆ ಮೌಲ್ಯ ಕಡಿಮೆ.

 ದಯವಿಟ್ಟು ಈ ಕೆಳಗಿನ ಸಲಹೆಗಳನ್ನು ಪರಿಗಣಿಸಿ:

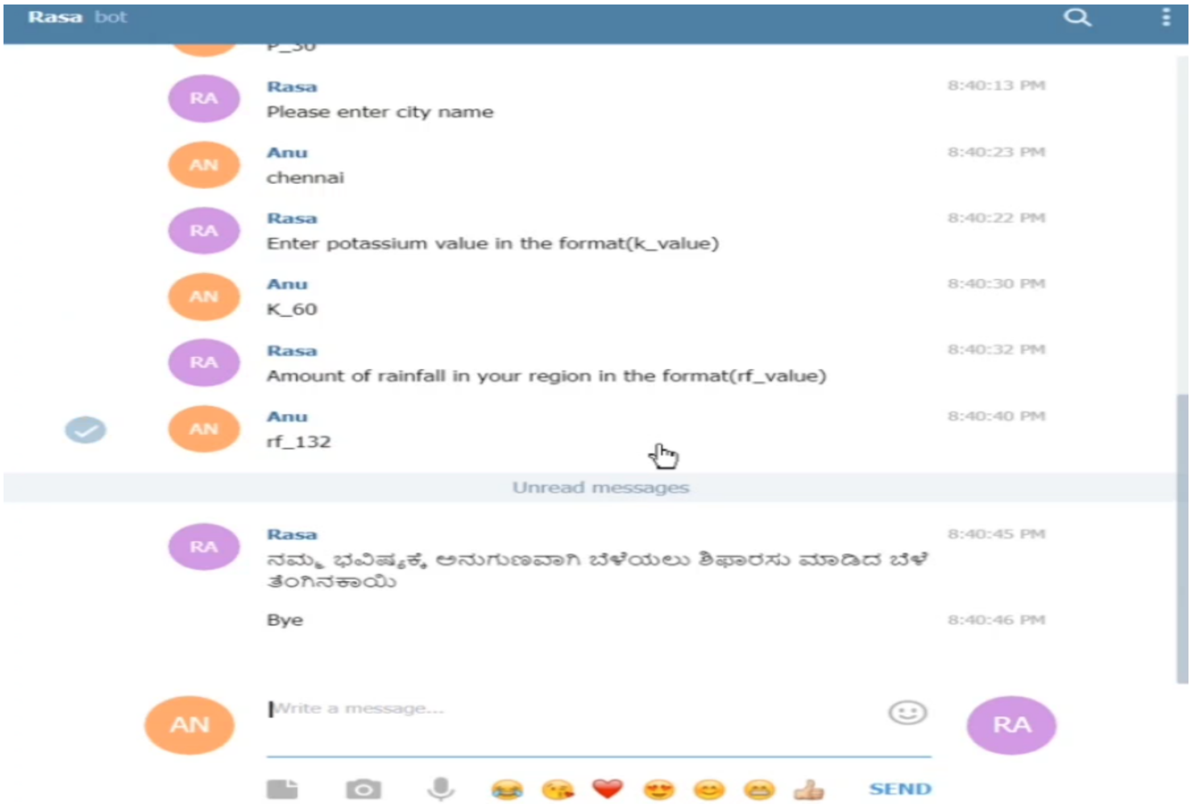
 1. ಮುರಿಕೇಟ್ ಆಫ್ ಪೊಟ್ಯಾಶ್ ಅಥವಾ ಸಲ್ಫೇಟ್ ಆಫ್ ಪೊಟ್ಯಾಶ್‌ನಲ್ಲಿ ಮಿಶ್ರಣ ಮಾಡಿ

 2. ಕೆಲ್ಪ್ meal ಟಿ ಅಥವಾ ಕಡಲಕಳೆ ಪ್ರಯತ್ನಿಸಿ

 3. ಸುಲ್-ಪೊ-ಮ್ಯಾಗ್ ಪ್ರಯತ್ನಿಸಿ

 4. ಬಾಳೆಹಣ್ಣಿನ ಸಿಪ್ಪೆಯನ್ನು ಮಣ್ಣಿನ ಮೇಲ್ಮೈಗಿಂತ ಒಂದು ಇಂಚು ಕೆಳಗೆ ಹೂತುಹಾಕಿ

 5. ಪೊಟ್ಯಾಶ್ ಗೊಬ್ಬರಗಳು ಹೆಚ್ಚಿನ ಮೌಲ್ಯದ ಪೊಟ್ಯಾಸಿಯಮ್ ಅನ್ನು ಹೊಂದಿರುವುದರಿಂದ ಅವುಗಳನ್ನು ಬಳಸಿ
8:36:05 PM
Bye
8:36:06 PM
AN
Write a message...
RA
SEND



[CROP PRICE VIDEO]

[FERTI INFO VIDEO]

[FERTI RECOMMENDATION VIDEO]

[CROP RECOMMENDATION VIDEO]