Pluto: AI-Based Meeting Scheduler

[1] Sridevi Tumula,[2] Sai Vinitha Yeggadi,[3] V. Indira Priyadarshini
[1]Associate Professor ,[2,3]Student
[1]Computer Science and Engineering,
[1]Chaitanya Bharathi Institute of Technology, Hyderabad, India

Abstract: Scheduling a meeting manually is a low level but is a time-consuming task. It cannot be done in a single go; instead, it takes multiple emails with several backs and forth negotiations over the date and time. To solve this problem and make this process automated, this scheduler is introduced. By conversing with this virtual assistant, the user can initiate a meeting by specifying the dayparts/time and set of participants for the meeting. This system's main aim is to schedule a meeting efficiently, i.e., without conflicts and with minimum user interactions. The system interprets user utterance using Intent Classifier followed by Name Entity Recognizer. Multi-Label Classifier maps initiator's utterance (example afternoon) to multiple possible time slots. After estimating time slots, Reinforcement Learning is used to select an agreeable timeslot. Based on the prior experience of scheduling meetings with different users in the system, the reinforcement learning-based bot attempts to choose slots with the maximum probability of participants agreeing. The agent will automatically schedule an appointment and tag the conference details to the user's calendar.

IndexTerms **– Agent , Scheduling,Process Automation,Virtual Assistant,Classifier,Reinforcement Learning.**

I. INTRODUCTION

One of the most frequently performed organizational tasks is scheduling meetings between employees across different designations and time zones. This scheduling of meetings is commonly carried out over an email or by human assistants, often involving back and forth negotiations over the meeting's real timeslot. This kind of mundane scheduling consumes a lot of time, effort and has much ambiguity. Automated workflow allows one to define their availability, removing the risk of double booking a time slot, ambiguity, and eliminating back and forth emails. Therefore a system is required, which interacts with personnel regarding the meeting scheduling and requires minimum user interaction. A study shows that if a person has ten appointments per day, it will take approximately 8 hours a week to schedule meetings alone, which is infertile. To solve this problem and to resourcefully utilize our time, automation is required. The main objective of this project is to minimize the number of interactions with the meeting participants over the negotiation of timeslot and automatically schedule a meeting at a given time slot and automatically carrying out the essential work like selecting a time slot and creating a meeting link by a machine.

II. METHODOLOGIES

User communicates with the bot using platforms like Telegram. The user's intention is identified from the user's utterance through the intent classifier.

Intent Classification:

Intent classification is a sub-task of NLU which determines which intent category the given intent belongs to. Intent refers to the goal the customer has in mind when typing in a question
or comment. For example, "Schedule meeting with Ram" , "Can you please schedule a meeting with Ram?" have the same meaning even though the expression is different, i.e., both belong to the same intent of "schedule_meeting" . Therefore to identify this, an intent bot uses deep learning and natural language processing to associate sentences with a particular intent automatically.

With sufficient amounts of training data, the intent classifier can correctly identify the intent of the user's utterance without human assistance, which makes the bot understand, interpret and respond faster and offers personalized service to the user. We propose a deep learning model for intent classification.

Entity Extraction:

Name Entity Recognition (NER) is a standard NLP problem that involves spotting named entities like people, places, organizations, etc., from a chunk of text and classifying them into a predefined set of categories. The entities identified from the utterance are a list of participants, daypart, date, and time. Entities are recognized from the utterance using deep convolutional neural networks with residual connections with word embedding strategy.

Multi-Label Classification:

As the user's utterance consists of vague timeslots instead of a specific time slot, time should be figured out from the user's utterance. Therefore We propose a multi-label learning approach to map sentences to multiple possible time slots. The approach utilizes a Long Short Term Memory (LSTM) based multi-label learning approach that utilizes independent loss functions over the output units and learns to map the sentences to correct time slots.

Q-Learning: Q-learning is a value-based learning algorithm. Value-based algorithms update the value function based on an equation (particularly Bellman equation).

The Q-function uses the Bellman equation and takes two inputs: state (s) and action (a). Initially, the Q-table has to be built.

There are n columns, where n= number of actions.

There are m rows, where m= number of states.

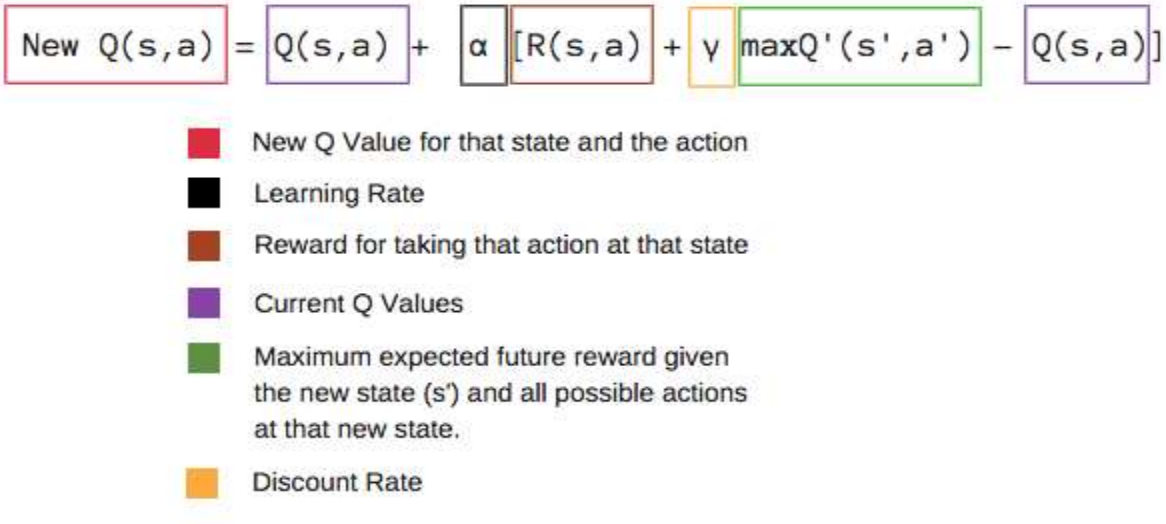

Figure 1: Q function

First, an action (a) in the state (s) is chosen based on the Q-Table. When it initially starts, every Q-value should be 0. During the process of exploration, the agent progressively becomes more confident in estimating the Q-values. We can now update the Q-values for being at the start and moving right using the Bellman equation. Now we have to take action and observe an outcome and reward. We need to update the function Q(s, a). This process is repeated again and again until the learning is stopped. In this way, the Q-Table is updated, and the value function Q is maximized. Here the Q(state, action) returns the expected future reward of that action, and Bot uses this Q-table to find them agreeable and best timeslot. This intent is further fed to an algorithm trained to map dayparts in a sentence to appropriate time slots. All the possible time slots are extracted then the probability of approval is calculated. This is multitask learning, where multiple tasks are resolved at the same time. The agreeable time slot is figured out by Reinforcement learning.

Reinforce:

Monte Carlo Policy gradient method works by choosing actions directly from a parameterized model, then updating the model's weights to nudge the following predictions towards higher expected returns. REINFORCE achieves this by collecting an entire trajectory then updating the policy weights in a Monte Carlo style. While keeping the bias unchanged, this tries to reduce the variance of gradient estimation. Here optimization of the optimal policy is performed. This tries to maximize the cumulative future reward by optimizing the policy function using gradient ascent. Using this, the bot tries to schedule meetings with maximum reward.

III RELATED WORK

Intent classification is one of the main tasks of a chatbot to respond accurately based on the context. Previous work on intent classification has mainly focused on the supervised learning method. Almost all standard approaches to classification have been applied in intent classification from SVM, Hidden Markov Models to Decision Trees, Bayesian Networks, and Rule-based approaches. These studies do not consider context information from the whole session-level. The main disadvantage of previous methods is their computational cost and high dependency, size of the training data in recognizing contexts within the same user utterance, and correctly identifying the user's intention. Recently deep learning is being used to solve this problem more efficiently.

Recently, few commercial applications came into the market to automate this process. There is still research going on to find efficient schedulers considering their personal, professional life. Different approaches like Astar, RAD were used to develop an optimal solution. Applying reinforcement learning is new to this field.

A policy-based gradient descent model is developed to schedule a meeting at a time slot that utilizes deep neural networks to approximate the policy function. The environment consists of the current occupancy of slots, designation of participants and the list of meetings to be scheduled. Given a state st, the policy determines which action the agent should perform to transform to state st+1 and receive reward rt from the environment. The policy is defined as a probability distribution over the actions π(s, a). This utilizes the method of policy gradients to learn a policy function that maximizes future reward.

$$\nabla_\theta E_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t] = E_{\pi_\theta}[\nabla \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)] \quad (1.1)$$

Where γ is the discount factor, the above function is utilized to update the policy function to gain maximum future reward. This policy's objective is to maximize the expected cumulative discounted reward.

for *each episode* **do**
  for *each timestep* **do**
    record $(s_1^i, a_1^i)(s_2^i, a_2^i)...(s_n^i, a_n^i)$
    compute benchmark & meetings_scheduled
    if *meetings_scheduled >= benchmark* **then**
      reward:= +1 $\forall\ (s^i, a^i)$
    else
      reward:= -1 $\forall\ (s^i, a^i)$
    end
    train using $(s_1^i, a_1^i, r_1^i)(s_2^i, a_2^i, r_2^i)...(s_n^i, a_n^i, r_n^i)$
  end
end

Figure 2: Reinforcement Learning Algorithm

This bot follows the above algorithm to schedule meeting. For each timestamp, it tries to schedule a meeting if the priority is greater than the benchmark; otherwise inserts it back into the waiting list.

The above-proposed system is not suitable for actual-world application as it tries to schedule all the meetings at once and does not have any natural language understanding capabilities. Therefore, we propose a system that can be integrated into the real world.

IV SYSTEM ARCHITECTURE

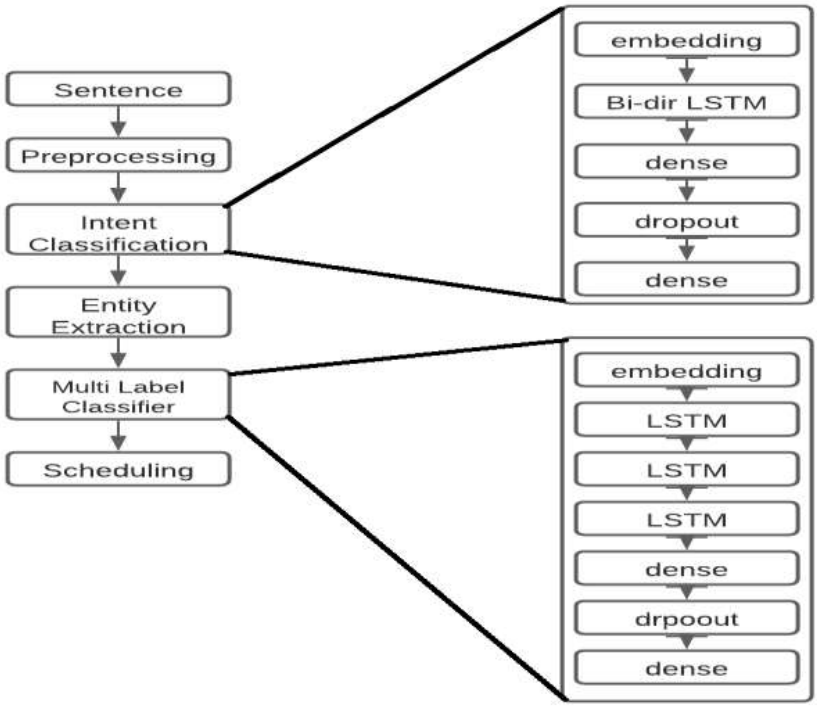

Figure 3: System Architecture

Intent Classification:

Intent Classification is the task of correctly labeling a natural language utterance from a predetermined set of intents. Intent refers to the goal the user has in mind when typing in a sentence or comment. For example, the model can learn that words such as schedule are often associated with the intent to schedule.meeting. Here, an LSTM based model is used to classify the intents.

Embedding Layer: A embedding W: words→Rn is a parameterized function mapping word sentences to high-dimensional vectors (perhaps 200 to 500 dimensions). It is the first layer of a network. It enables us to convert each word into a fixed-length vector to defined size.

It has three arguments: vocabulary size, output_dimention and input_length. Suppose we define an embedding layer with a vocabulary size of 146, a vector space of 256 dimensions to which words will be embedded and an input sequence with 15 words each. The output of that layer is a 2D vector with one embedding for each word in the input sequence of words.

LSTM: LSTM is a special kind of RNN capable of learning long-term dependency problems. They are designed to avoid the long-term dependency problem. LSTM has a chain-like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four interacting layers.

```
model.add(Embedding(vocab_size, 256, input_length = max_length))
```

Figure 4: Embedding layer arguments

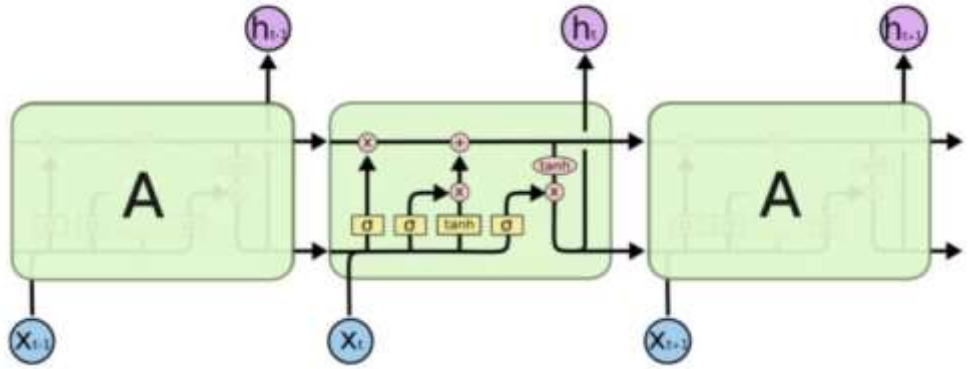

Figure 5: LSTM Interacting Layers

The key to LSTM is the cell state, the horizontal line running through the top of the diagram. The LSTM can remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to let information through optionally. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.
The first step in our LSTM is to decide what information we will throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at ht 1 and xt and outputs a number between 0 and 1 for each cell state Ct-1.

A 1 represents "completely keep this", while a 0 represents "completely get rid of this."The next step is to decide what new information we are going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we will update. Next, a tanh layer creates a vector of new candidate values, C~t, that could be added to the state. In the next step, we will combine these two to create an update to the state.Now we update the old cell state, Ct−1, into the new cell state Ct. The previous steps already decided what to do, so we multiply the old state by ft, forgetting the things we decided to forget earlier. Then we add it ∗ C~t. These are the new candidate values, scaled by how much we decided to update each state value.

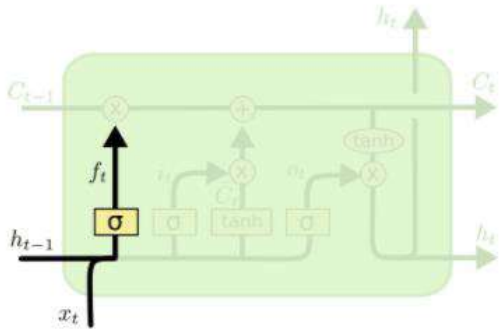

Figure 6: Forget Gate

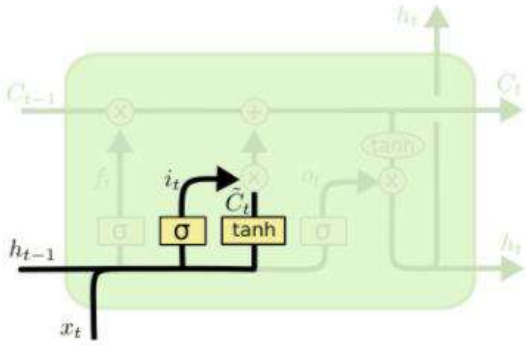

Figure 7: input gate layer and tanh layer

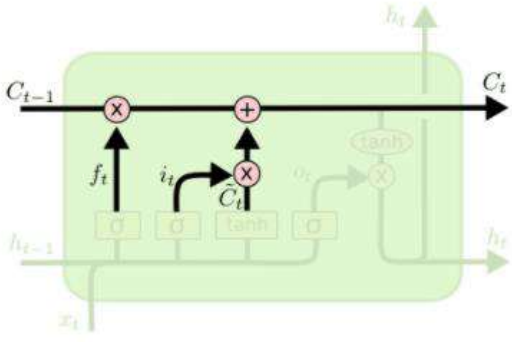

Figure 8: Update of old cell state

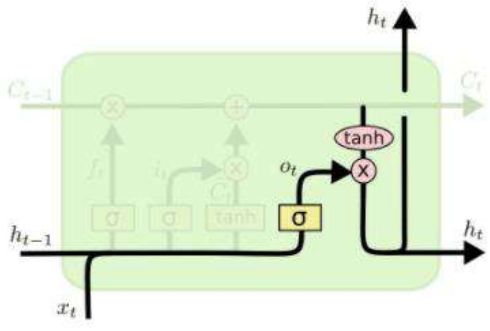

Figure 9: tanh and sigmoid gate for output

Finally, we need to decide what we are going to output. This output will be based on our cell state but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we are going to output. Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate so that we only output the parts we decided to.
Bidirectional LSTM: Bidirectional LSTMs are an extension of traditional LSTMs that can improve model performance. Bidirectional LSTMs train two instead of one LSTMs on the input sequence. The first on the input sequence as-is and the second

on a reversed copy of the input sequence. This can provide additional context to the network and result in faster and even fuller learning on the problem.

Entity Extraction: Entity extraction is a text analysis technique that uses Natural Language Processing (NLP) to pull out specific data automatically from unstructured text and classifies it according to predefined categories.

These categories are named entities, the words or phrases that represent a noun. This includes proper names and numerical expressions of time or quantity, such as phone numbers, monetary values, or dates. The entities to be extracted in this application are the Names of the participants, day, daypart, duration. For example: "Schedule meeting with Mikasa today afternoon for 30 minutes" The entities here are Mikasa - Name, today - day, afternoon - daypart, 30 minutes - duration.

Each word represents a token: "Schedule a meeting" is a string of three tokens representing one entity. Inside-outside-beginning tagging is a common way of indicating where entities begin and end. The second step requires the creation of entity categories.

```
[('Schedule meeting with Vinitha today afternoon for 30 minutes ', {'entities': [(23, 30, 'NAME'), (31, 37, 'DATE'), (3
8, 48, 'DAYPART'), (54, 56, 'DURATION')]}), ('Schedule meeting with Indira today evening for 20 minutes ', {'entities':
[(23, 29, 'NAME'), (30, 36, 'DATE'), (37, 44, 'DAYPART'), (50, 52, 'DURATION')]}), ('Can you schedule a meeting for to
day evening with Vinitha 45 minutes ', {'entities': [(37, 43, 'DATE'), (44, 52, 'DAYPART'), (59, 66, 'NAME'), (67, 69, 'D
URATION')]}), ('Please schedule meeting with Indira today morning ', {'entities': [(30, 36, 'NAME'), (37, 43, 'DATE'), (4
4, 51, 'DURATION')]}), ('Schedule meeting with Mikasa tomorrow evening for 20 minutes ', {'entities': [(23, 29, 'NAME'),
(30, 38, 'DATE'), (39, 46, 'DAYPART'), (52, 54, 'DURATION')]}), ('Schedule meeting with Vinitha today late-morning for 4
0 minutes ', {'entities': [(23, 30, 'NAME'), (31, 37, 'DATE'), (38, 51, 'DAYPART'), (57, 59, 'DURATION')]}), ('Schedule meet
ing with Vinitha today late-evening for 40 minutes ', {'entities': [(23, 30, 'NAME'), (31, 37, 'DATE'), (38, 50, 'DAYPAR
T'), (56, 58, 'DURATION')]}), ('Can you schedule a meeting for today late-morning with Vinitha 45 minutes ', {'enti
ties': [(37, 43, 'DATE'), (44, 57, 'DAYPART'), (64, 71, 'NAME'), (72, 74, 'DURATION')]}), ('Can you schedule a meeting
for today late-afternoon with Vinitha 45 minutes ', {'entities': [(37, 43, 'DATE'), (44, 59, 'DAYPART'), (66, 73, 'NAM
E'), (74, 76, 'DURATION')]}), ('Schedule meeting with Mikasa tomorrow afternoon for 20 minutes ', {'entities': [(23, 29,
'NAME'), (30, 38, 'DATE'), (39, 49, 'DAYPART'), (55, 57, 'DURATION')]}), ('Schedule meeting with Mikasa tomorrow evening fo
r 20 minutes ', {'entities': [(23, 29, 'NAME'), (30, 38, 'DATE'), (39, 46, 'DAYPART'), (52, 54, 'DURATION')]})]
```

Figure 10:Spacy input list of tuples

Time slot Extraction: We propose a multi-label learning approach to map initiator utterances to multiple possible time slots. In multi-label classification, the training set is composed of instances, each associated with a set of labels, and the task is to predict the label sets of unseen instances through analyzing training instances with known label sets. Based on the meeting traffic and the prior experience of scheduling meetings with different users in the system, the
agent attempts to choose slots. It estimates the maximum probability of the participants agreeing.

The model has an embedding layer followed by stacked LSTM layers. The padded dayparts sequence and one hot encoded timeslots are fed to the embedding layer. The embedding layer creates a dense vector of real numbers. This is fed to a series of LSTM layers. The upper LSTM layer provides a sequence output, and the last LSTM layer creates a single vector containing information of the entire sequence. This is followed by a dense layer and dropout layer. The last layer consists of a dense layer with a softmax activation function.

Scheduling: The environment automatically decides upon a meeting and a slot. The agent has to decide between two possible actions, i.e. whether or not to schedule the selected meeting at the chosen slot by issuing a request to the meeting participants. If the agent decides to schedule, a dialogue is initiated wherein the agent must request the participants to attend the meeting at the chosen slot. The reply is processed to determine whether the user agreed. The resulting experiences are rewarded with both immediate and delayed rewards to balance the tradeoff between scheduling effectively avoiding unnecessary meeting requests.

Q- learning selects the best rewarding timeslot and checks for the existing events. If the bot tries to schedule on a timeslot with existing events, there is a penalty to the bot. Otherwise the bot is rewarded. Another reinforcement learning implemented is policy-based reinforcement learning which has a policy.

Q-learning is a model-free reinforcement learning algorithm. Q-learning is a values-based learning algorithm. Value-based algorithms update the value function based on an equation (particularly Bellman equation). The 'Q' in Q-learning stands for quality. Quality here represents how beneficial a given action is in gaining some future reward.

1. First, the Q-table has to be built. There are n columns, where n= number of actions. There are m rows, where m= number of states.
2. Choosing the action
3. The combination of steps 2 and 3 is performed for an indefinite amount of time. These steps run until the time training is stopped or when the training loop stops as defined in the code. First, an action (a) in the state (s) is chosen based on the Q-Table. When the episode initially starts, every Q-value should be 0. Then, update the Q-values for being at the start and moving right using the Bellman equation.
4. Now we have taken action and observed an outcome and reward.
5. We need to update the function Q(s, a). This process is repeated again and again until the learning is stopped. In this way, the Q-Table is updated, and the value function Q is maximized. Here the Q(state, action) returns the expected future reward of that action at that state.

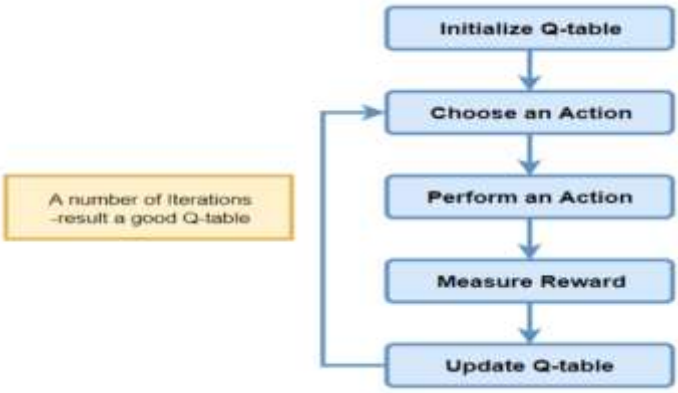

Figure 11: Q Learning flowchart

If the participants agree to the timeslot selected, then the bot is rewarded. This reward is updated in the q-table. This is done based on a Q-learning algorithm. Q learning updates the q table using the formula shown in figure 13.

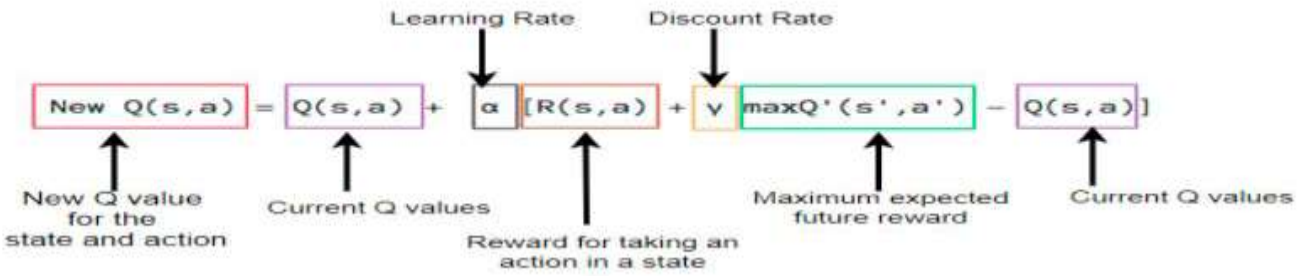

Figure 12: Q-function

Monte-Carlo Policy-based gradient descent:

Initialize a policy and a reward array to store the experience from the episode. Following the policy, an episode is generated, an action is chosen, rewards are generated based on that action. Based on the output, the policy is improved.

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$

Figure 13: Monte Carlo policy- gradient control

The objective here is to learn a policy that maximizes the cumulative future reward. Therefore, we will optimize the policy by taking gradient ascent with the partial derivative w.r.t ⊖. After successfully scheduling a meeting, ⊖ is updated with future reward.

Here bot learns the policy function that maps state to action. This consists of a simple policy; the gradient is found using a derivative of softmax function, full jacobian is calculated. Then the weights are updated by log policy gradient times future reward. This is updated after each episode.

DATA SETS

Schedule a meeting today early morning with Vivek for 25 minutes	schedule.meeting
Schedule a meeting today late morning with Vivek for 30 minutes	schedule.meeting
Schedule a meeting today afternoon with Vivek for 35 minutes	schedule.meeting
Schedule a meeting today evening with Vivek for 40 minutes	schedule.meeting
Schedule a meeting today late evening with Vivek for 45 minutes	schedule.meeting
Schedule a meeting tomorrow morning with Vivek for 20 minutes	schedule.meeting
Okay schedule	schedule.confirm
Yes i am free	schedule.confirm
I can attend	schedule.confirm
Schedule it	schedule.confirm
Okay	schedule.confirm
Ok	schedule.confirm
Schedule	schedule.confirm
Yup	schedule.confirm
Okay schedule	schedule.confirm
Yes i am free	schedule.confirm
I can attend	schedule.confirm
Schedule it	schedule.confirm
Okay	schedule.confirm
Ok	schedule.confirm
Schedule	schedule.confirm
Yup	schedule.confirm
No i cannot	schedule.cancel
No	schedule.cancel
Nope	schedule.cancel
Do not schedule	schedule.cancel
Don't schedule	schedule.cancel

Figure 14: Dataset snippet of intent classifier

Dataset for intent classification

This dataset consists of queries and their corresponding intent class. User can express his intention to schedule a meeting with any syntax structure, meaning he need not follow specific order but, the semantics is the same for all the queries of an intent class. For example, The user's utterance might be "Can you schedule a meeting for tomorrow morning with Mikasa" or "Please schedule a meeting with Indira this evening" or "Meeting with Mikasa evening for 20 min". These sentences are expressed differently, but the meaning behind them is the same, i.e., schedule a meeting.

This dataset consists of 9 different intent classes. They are:

'Schedule.meeting': This consists of queries to schedule a meeting with the given participants, given duration on a particular day on a specific daypart.

Example: Schedule meeting with Mikasa evening for 20 min

'Schedule.confirm': This consists of confirmation messages accepting the meeting's timeslot. Example: Okay schedule

'Schedule.later': This rejects the current generated timeslot and asks the bot to schedule later. Example: Postpone it.

'Schedule.cancel': Cancel the meeting Example: I am not free

'Commonq.not_giving': User is not interested

'commonq.name': Details of bot Example: Are you there?

'commonq.just_details': User's details Example : Show my phone number 'commonq.bot': Common questions related to bot.

Example: This is a machine? 'commonq.how': Details related to the bot. Example: How are you doing?

Entity extraction dataset

Figure 15 is the snippet of the entity extraction dataset. This dataset consists of sentences in which each word is mapped to the corresponding entity category. The required entities are Daypart, name, day and duration. Therefore their corresponding words are mapped. As the remaining data is not necessary for this application, the rest of the words are mapped to O. These sentences are separated by "." (dot). The entity extractor will assess the entities present in the user's utterance. This dataset consists of approximately 100 sentences, with each word mapped to corresponding entity names.

Schedule	O
meeting	O
with	O
Indira	NAME
today	DATE
evening	DAYPART
for	O
20	DURATION
minutes	O
.	O
Can	O
you	O
schedule	O
a	O
meeting	O
for	O
today	DATE
evening	DAYPART
with	O
Vinitha	NAME
45	DURATION

Figure 15: Dataset snippet of an entity extractor

early-morning	7
morning	10
late-morning	11:30
early-morning	8
late-morning	10
afternoon	12
early-morning	07:30
evening	15
late-morning	11:30
late-evening	20
early-morning	8
morning	10
late-morning	12
evening	15:30
afternoon	13
evening	16
late-evening	19
early-morning	9
morning	9
late-morning	11
afternoon	14
early-morning	07:30

Figure 16: Dataset snippet to discover timeslot

III RESULT

Table 1: Result of Intent Classifier

Model	Activation function	Training Accuracy	Testing Accuracy
LSTM	Softmax	100%	72%
Bi-LSTM	Softmax	100%	88%

Table 2: Result of Multi-Label Classifier

Model	Activation function	Precision	Recall
Stacked LSTM	Softmax	97.3%	97.3%
Stacked Bi-LSTM	Softmax	97%	96%
Stacked LSTM	Sigmoid	98.4%	98.4%
Stacked Bi-LSTM	Sigmoid	99.8%	99.9%

User communicates with the system through telegram. As the user sends the sentence/utterance to the telegram(bot) saying" schedule meeting with Vinitha today

afternoon for 30 minutes" it finds its intents, entities, and best time slot from that sentence and replies saying "will schedule a meeting."

Figure 17: Screenshot of telegram bot

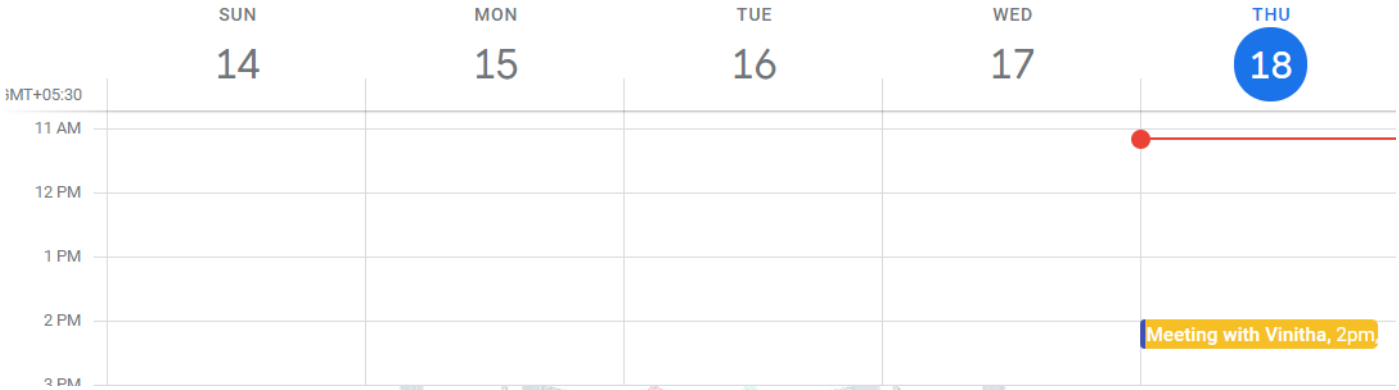

Figure 18: Screenshot of user's calendar

From the extracted entities and user-preferred timeslot, the details are added to the calendars based on meeting day, date, and time.

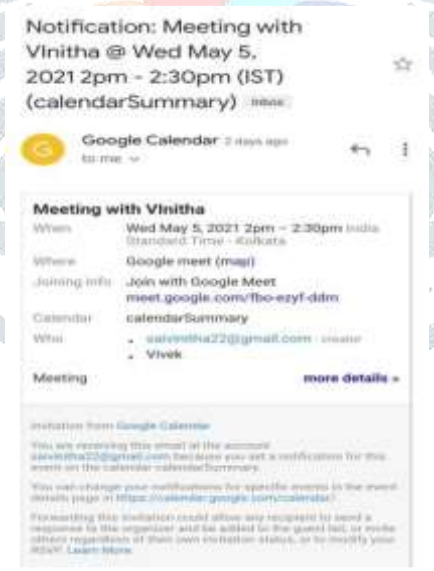

Figure 19: Screenshot of email notification

Email notification: The participants receive an email notification including all the information like when the meeting is, where on what platform, meeting link, who is hosting, and all the participants list in that email also, giving a time-to-time reminder to all the participants about the meeting.

CONCLUSIONS

We have successfully developed a meeting bot that can schedule meetings through user utterances/sentences interacting from external platforms like telegram. Furthermore, an intent classifier to classify the user's intents gives to the bot as an utterance/sentence. Also, an Entity Extractor to find the entities in the sentence like name, date, daypart, and time. A multi-label classification model to convert English phrases that have time information into slots was employed with separate output loss functions for each time slot. A model using reinforcement learning is trained to schedule them efficiently. The model can adapt to new situations with varying meeting arrival rates, and the performance of the model is compared with standard schedulers. We have also shown that the RL agent can adapt to user preferences and schedule meetings accordingly, and can also change its policy.

Limitations

The bot accesses the participants' calendars using their credentials and tokens; these tokens of the google calendar expire after a certain period depending upon the last accessed time. Therefore, the tokens have to be regenerated. This might be an issue for an organization as the bot needs to monitor the token's expiry time and request for authentication of the calendars.

Currently, this bot uses Gmail to communicate with the users. Peer-to-peer interaction is not possible here. This will limit the interaction of the participants while scheduling the meeting.

Currently, this bot waits 5 minutes for the reply. However, in real-time, a 5-minute lifespan is very less as the person might be busy and may not respond quickly. Therefore bot should wail some more time depending on the organization and the time the usually take to reply.

Future Work

As every organization has its organization-specific communicating platform, this can be integrated into that specific platform. For each request, a temporary group can be created to communicate with the users, and the bot can also monitor the views of all the participants. Depending upon the views, the bot tries to schedule meetings instead of one-on

one communication with the bot.

This bot can be improved to solve other problems in the organization instead of limiting it to meeting scheduling. This bot can also be extended to plan office trips. Another extension can be scheduling personal meetings with the bot and planning personal trips using this bot.

Scope of the project

Meeting scheduling requires users to spend countless hours unproductive and mismanaged meetings, but the greater crime is all the wasted time the user spends scheduling that meeting. When a user tries to schedule a meeting, it leads to an excessively long email chain trying to pinpoint the best time and date for all the participants. The endless back and forth communication steps like receiving a calendar, setting up a call-in number, adding a meeting

to the calendar, and inviting all the necessary attendees are drawbacks of physical scheduling. After confirmation, he has to send meeting details to all the participants with that information. Automatic meeting scheduling allows us to define availability, remove the risk of double booking a slot and eliminate the back and forths. The main aim is to reduce this back and forth negotiation of timeslot and integrate this bot with real-time applications like telegram.

REFERENCES

[1]. Vishwanath D, Lovekesh Vig, Gautam Shroff & Puneet Agarwal. 2018. MEETING BOT: Reinforcement Learning for Dialogue Based Meeting Scheduling. arXiv:1910.11470

[2]. Ruder. 2017. An overview of multi-task learning in deep neural networks.arXiv:1706.05098

[3]. S. M. M. Hossain and E. M. Shakshuki,2013. A Deliberative Agent for Meeting Scheduling. IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), 2013, pp. 732-739.

[4]. Leonardo Garrido-Luna and Katia Sycara. 2018. Towards a Totally Distributed Meeting Scheduling System. Annual German Conference on Artificial Intelligence: Advances in Artificial Intelligence. J. Intell. Robotic Syst. Journal

[5]. Vikas Yadav University of Arizona, Steven Bethard. 2018.A Survey on Recent Advances in Named Entity Recognition from Deep Learning models. arXiv:1910.11470

[6]. S. Ravuri and A. Stoicke. A comparative study of neural network models for lexical intent classification. 2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), 2015, pp. 368-374

[7]. J. Schuurmans and F. Frasincar,2020. Intent Classification for Dialogue Utterances in IEEE Intelligent Systems, vol. 35, no. 1, pp. 82-88

[8]. Kian L. Pokorny, Ryan E. Vincent2019. Multiple constraint satisfaction problems using the a-star (a*) search algorithm: classroom scheduling with preferences. Journal of Computing Sciences in College, pp. 152-159

[9]. E. R. Subhiyakto and Y. P. Astuti. 2019. Design and Development Meeting Schedule Management Application using the RAD Method. 2019 International Conference of Artificial Intelligence and Information Technology (ICAIIT), 2019, pp. 60-64

[10]. Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning Journal pp. 229-256