Final Project

Large Language Model–based Agents

Group 10

Maynard Smid

Prashant Yadava

06.01.2024

Introduction

In the final project we implemented an auction platform where artificial intelligence agents built upon large language models were instantiated for playing the roles of auctioneer and bidders. The agents can understand and respond to prompts presented in natural language resembling human communication.

Our implementation has several bidders submitting their bid for an auction conducted by an Auctioneer, where the winner is chosen as per the Vickery auction mechanism (The bidder bidding the highest bid wins and pays the 2nd highest bid amount).

In addition to the mandatory part of the project, we also implemented the challenge where multiple auctions were required to be conducted.

How to run

Google Colab

We used Google Colab as our implementation and execution environment, but in principle, any local deployment of interactive python notebook could be used to run our implementation. The environment has all the necessary tools and packages already built-in and using them is as easy as invoking the correct package or model by name.

The implementation of the required component is found here.

The implementation of the challenge is found here.

OpenAI ChatGPT Model

We used OpenAI's ChatGPT3.5 turbo model. The model's usage is offered as a paid credit-based service by OpenAI.

Microsoft Autogen

AutoGen is an open-source framework that allows building LLM applications via multiple agents that can converse with each other to accomplish tasks. For the project, we used the Autogen instance that is offered for free as part of the Colab environment.

Assignment: Mandatory Part

Agents

In the mandatory part of the assignment, we conducted a single round of Vickery auction. The various agents participating in the auction, namely the bidders and auctioneer were implemented as LLM agents by instantiating respective Agent classes from Autogen.

The auction itself was implemented using autogen agents which can perform tasks via automated chat. Here we had agents communicating back and forth via messages which are based on prompts which are carefully designed to implement the FIPA communication protocol.

Auctioneer

The auctioneer agent, which is responsible for conducting the auctions, is based on Autogen's AssistantAgent class. Below screenshot shows the prompt used to design this class which determines its role and responsibility in the auction setting.

```
auction_agent = autogen.ConversableAgent(
    name="AUCTIONEER",
    system_message="""You are an AI-powered Auctioneer.
You cooridinate a Vickery auction.
You receive bids from the different bidders.
The winner is the agent with the higest bid and the amount they must pay is the second highest bid.
when asked who won the auction return the answer in this format: Winner_name, int(amount).
If there is a problem just return the word error and explain why there was an error.""",
    llm_config=llm_config,
    max_consecutive_auto_reply = 3
)
```

Bidder

Next, we have the bidders implemented as Autogen's AssistantAgent class. The prompt below shows how they are customized and configured to submit bids for the Vickery auction conducted by the auctioneer. The number of bidders is dynamic and can be changed by changing the 'numbidders' variable.

```
#creates number of bidders specified by user and gives them instructions
for i in range(numbidders):
  bidder_names.append("bidder" + str(i))
  bidders.append(autogen.AssistantAgent(
    name=bidder_names[i],
    system_message = """Your name is bidder""" + str(i) + """and you are participating in a vickery auction as a bidder.
The auction is conducted by {auction_agent}.
You should start with a randomly generated bid value in the range of 500-1000 and send it to the auctioneer when requested.
Do not include extra chit-chat
""",
    llm_config=llm_config,
    max_consecutive_auto_reply = 1
  ))
```

User Proxy

In addition, we also have a human-proxy user-agent which allows us to have an interface to a human's involvement in the auction as needed. Autogen provides the class UserAgentProxy for this purpose.

```
user_proxy = autogen.UserProxyAgent(
    name="User_proxy",
    system_message="A human admin.",
    human_input_mode="NEVER"
)
```

We are using an instance of this agent for a basic use, namely, to let the Auctioneer report back the auction's outcome.

Implementation

The Auctioneer kickstarts the auction by declaring the start of auction process of an item (for example, a valuable painting) and requesting bids from the agents. Upon receiving the message, each bidder proceeds to respond with its bid. We decided to choose a random number between a pre-defined range for the bid, as this was sufficient to fulfil the requirements of the assignment, but ideally this would be determined based on some utility function.

The auctioneer selects the highest bidding agent as the winner and informs its dues to the tune of the 2nd highest bid received. All other agents are informed of their failure to win.

Table 1 below presents the output from a sample execution showing the auction in action. As can be seen, Bidder2 with its bid amount of $845 won the auction and must pay $750 which is the second highest bid.

Interaction was implemented using FIPA for communication, and for this purpose we are again using a slightly adapted version of the 'FIPA English Action Interaction Protocol' [1].

```
AUCTIONEER (to bidder1):

 I'm starting a new Vickery auction for a valuable painting. Please send me your
bid and your name.

 --------------------------------------------------------------------------------

bidder1 (to AUCTIONEER):

 My bid is 750. My name is bidder1.
```

```
 --------------------------------------------------------------------------------

AUCTIONEER (to bidder2):

 I'm starting a new Vickery auction for a valuable painting. Please send me your
bid and your name.

 --------------------------------------------------------------------------------

bidder2 (to AUCTIONEER):

 My bid is 845 and my name is bidder2.

--------------------------------------------------------------------------------

User_proxy (to AUCTIONEER):

 The bids received were {'content': 'My bid is 750. My name is bidder1.',
'role': 'assistant'}{'content': 'My bid is 845 and my name is bidder2.', 'role':
'assistant'}{'content': '500', 'role': 'assistant'}Who had the highest bid? And
what was the second highest bid? Return the winner and the second highest bid
which is the amount they'll need to pay in this format: (Winner_name,
int(amount_to_pay)). Provide only the answer

 --------------------------------------------------------------------------------

 >>>>>>>> USING AUTO REPLY...

AUCTIONEER (to User_proxy):

 (bidder2, 750)

 --------------------------------------------------------------------------------

AUCTIONEER (to bidder2):

 You won the auction. You must pay $750

 --------------------------------------------------------------------------------

AUCTIONEER (to bidder1):

 The auction is over. You have not won

 --------------------------------------------------------------------------------

AUCTIONEER (to bidder3):

The auction is over. You have not won

--------------------------------------------------------------------------------
```

[1]

Assignment: Challenge 1

For the challenge part of the assignment, we were required to implement simultaneous auctions of multiple items. We implemented 3 auction categories, namely “paintings”, “statues”, and “jewellery”, each managed by an auctioneer. We have decided to again implement Vickery auction in this part of the assignment. The challenge requires using FIPA for communication, and for this purpose we are again using a slightly adapted version of the ‘FIPA English Action Interaction Protocol’ [1] and also uses the ‘FIPA Propose Interaction Protocol’ [2] for determining the auction each bidder is interested.

Agents:

Auctioneer

We have 3 auctioneers, each responsible for one category of item. Each is tasked with determining the winner based on the bids received for its own category and respond with the winner and the payable bid (again, as per Vickery auction, the winner pays the 2nd highest bid and not its own bidding amount).

User Proxy

In this part, we used the user-proxy as the initiator of the multi-item auction. It informs the start of the auction to the participants and shares the category of items in the auctions and invites the bidders to choose which auction they’d like to join.

Bidders

The bidders can either choose to participate in the auction or not. If they do participate, they are required to indicate the category of the item they are interested in. In the prompt chosen by us for the Bidders, we instruct them to select a category at random.

Implementation

The implementation of the challenge differs from the mandatory part by having multiple items being auctioned at a time with different bidders bidding on only their own choice of item. The category selection is done using the ‘FIPA Propose Interaction Protocol’, where the user-proxy agent proposes categories for the bidders and the bidders accept a category or refuse. The bidding for the auction is again done as per a single round ‘FIPA English Action Interaction Protocol’, where after the initial start-of-auction message from the initiator from the User-Proxy, the Bidders respond with a proposal for their item (category) of choice, if they decide to enter the auction. Using this communication protocol for a Vickery auction leads to a single round of the bid offer from the Bidders as there is no rejection of proposal from the auctioneer.

Results

Below the output shows the initial broadcast message from the user-agent proxy sent to all the bidders. Based on the response from the individual bidders, the bidders are added to the respective list of items.

```
User_proxy (to bidder0):

 The categories for the auction are ['paintings', 'statues', 'jewellery'].
Return your name and the category you are interested in?

 --------------------------------------------------------------------------------

bidder0 (to User_proxy):

 My name is bidder0 and I am interested in the category of paintings.

 --------------------------------------------------------------------------------

User_proxy (to bidder1):

The categories for the auction are ['paintings', 'statues', 'jewellery']. Return
your name and the category you are interested in?

--------------------------------------------------------------------------------

bidder1 (to User_proxy):

My name is bidder1 and I am interested in the category of paintings.

...

...

...

User_proxy (to bidder24):

 The categories for the auction are ['paintings', 'statues', 'jewellery'].
Return your name and the category you are interested in?

 --------------------------------------------------------------------------------

bidder24 (to User_proxy):

 My name is bidder24 and I am interested in the category of jewellery.
```

Next, the auctioneer responsible for a particular category of item initiates the communication with only the bidders who had earlier shown interest in that category. The winner and the respective payable winning bid are determined as per Vickery auction. Below we present the output of message exchange between bidders the respective Auctioneer. First the Painting auctioneer:

```
paintings_AUCTIONEER (to bidder0):

Please send me your bid and your name.

--------------------------------------------------------------------------------

bidder0 (to paintings_AUCTIONEER):

My bid is 827 and my name is bidder0.

--------------------------------------------------------------------------------

paintings_AUCTIONEER (to bidder1):

Please send me your bid and your name.

--------------------------------------------------------------------------------

bidder1 (to paintings_AUCTIONEER):

My bid is 750 and my name is bidder1.

..

..

..

--------------------------------------------------------------------------------
paintings_AUCTIONEER (to bidder23):

Please send me your bid and your name.

--------------------------------------------------------------------------------
bidder23 (to paintings_AUCTIONEER):

My name is bidder23, and my bid is 750.
```

Next, the jewellery auctioneer:

```
--------------------------------------------------------------------------------

jewellery_AUCTIONEER (to bidder3):

Please send me your bid and your name.

--------------------------------------------------------------------------------

bidder3 (to jewellery_AUCTIONEER):

My name is bidder3. My bid is 850.

--------------------------------------------------------------------------------

jewellery_AUCTIONEER (to bidder4):

Please send me your bid and your name.

--------------------------------------------------------------------------------

bidder4 (to jewellery_AUCTIONEER)

My bid is 750 and my name is bidder4.

...

...

...
```

Interestingly, no interest was shown in the category Statues from any bidders. There could be several interesting reasons explaining it, but as it might involve speculation, we shall refrain from it.

After bids have been submitted by each bidder participating in the auction, the user proxy asks each auctioneer to determine the winner and inform each participant of their winning status.

```
The bids received were [{'content': 'My bid is 827 and my name is
bidder0.', 'role': 'assistant'}, ....{'content': 'My name is bidder23,
and my bid is 750.', 'role': 'assistant'}]

Return the bidder who placed the highest bid and the value of the second
highest bid amount in this format: (Winner_name, int(amount_to_pay)).
Provide only the answer

 --------------------------------------------------------------------------
---------

paintings_AUCTIONEER (to User_proxy):

 (bidder9, 827)

 --------------------------------------------------------------------------
--------

(bidder9, 827)

paintings_AUCTIONEER (to bidder9):

 You won the auction for paintings. You must pay $827

 --------------------------------------------------------------------------
---------

paintings_AUCTIONEER (to bidder0):

The auction for paintings is over. You have not won

--------------------------------------------------------------------------
--------



...

...

...

--------------------------------------------------------------------------
```

```
User_proxy (to statues_AUCTIONEER):

The bids received were []

Return the bidder who placed the highest bid and the value of the second
highest bid amount in this format: (Winner_name, int(amount_to_pay)).
Provide only the answer

--------------------------------------------------------------------------------

statues_AUCTIONEER (to User_proxy)

error, Insufficient bids received.

--------------------------------------------------------------------------------

There was an error, please try again

User_proxy (to jewellery_AUCTIONEER):

The bids received were [{'content': 'My name is bidder3. My bid is
850.', 'role': 'assistant'}, ... {'content': 'My name is bidder24 and my
bid is 750.', 'role': 'assistant'}]

Return the bidder who placed the highest bid and the value of the second
highest bid amount in this format: (Winner_name, int(amount_to_pay)).
Provide only the answer

--------------------------------------------------------------------------------

jewellery_AUCTIONEER (to User_proxy):

(bidder20, 850)

--------------------------------------------------------------------------------

(bidder20, 850)

jewellery_AUCTIONEER (to bidder20):

You won the auction for jewellery. You must pay $850

--------------------------------------------------------------------------------

...

--------------------------------------------------------------------------------

jewellery_AUCTIONEER (to bidder24):

The auction for jewellery is over. You have not won

--------------------------------------------------------------------------------
```

It is worth mentioning here that the communication protocol ideally allows for multiple iterations of proposals until there are no agents willing to offer a higher price than the current highest bid.

Conclusion

LLMs, like ChatGPT, used in this project have been trained on massive amount of text data and can leverage deep learning to capture and replicate complex patterns in language. In this project, through carefully designed prompts, we demonstrated how intelligent agents built on such LLM can be trained to excel in specialized tasks such as auctions. Frameworks such as Autogen, provide a very flexible yet powerful mechanism to have multiple heterogenous type of agents collaborate on solving complex problems together.

One of the most challenging aspects of this assignment was making the agents communicate consistently and reliably. The results are often indeterministic and the output varies constantly without changing prompts or communication. We found it was nearly impossible for them to communicate using the same protocol reliably, so we implemented the FIPA protocols using human language, as this is what the agents are designed to do. While they still don't often respond in the same format, the results can be parsed and separated enough for the other agents to interpret them.

References

1. FIPA English Action Interaction Protocol
   http://www.fipa.org/specs/fipa00031/XC00031F.html
2. FIPA Propose Interaction Protocol
   http://www.fipa.org/specs/fipa00036/XC00036F.html