ChatGPT for NLP

March 12, 2024

1 Assignment 2 - Use and Discuss ChatGPT

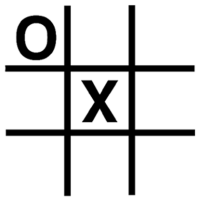
OpenAI developed ChatGPT, and by doing so, ushered the big-tech LLM race

This assignment analyzes ChatGPT, a state-of-the-art natural language processing model developed by OpenAI. By posing a series of questions and tasks to ChatGPT, this assignment aims to explore its strengths and weaknesses, shedding light on its performance in various scenarios.

Matthew Acs

2 Introduction

This assignment will take a closer look at ChatGPT as a natural language processing system. ChatGPT will be asked to complete 10 tasks related to the goal of creating a tic tac toe game using python on Google Colab. These tasks will allow us to evaluate ChatGPT, discover its advantages and limitations, and dive further into the technical reasons behind its shortcomings.



To begin let's imagine that we are lazy programmers who want to create a tic tac toe game. Our favorite programming language is python, and we use Google Colab hosted Jupyter notebooks because we are addicted to cloud computing (thanks big tech). Our first goal is simply to be able to play tic tac toe. Our second goal is for it to look good, and then we will want to add as many quality-of-life and aesthetic feature as we can. Finally, not only are we lazy, but we are also impatient, so we will only give ChatGPT 10 interactions to provide the final product. Let's see what ChatGPT can do!

3 Prompt 1

Me:

Create a tic tac toe game using python and google colab

ChatGPT:

Sure, here's a simple implementation of Tic Tac Toe in Python that you can run in Google Colab. To run this code in Google Colab, copy the code into a code cell in a Google Colab notebook. Run the cell. Follow the instructions printed in the notebook to play the game.

[1]:
```
2def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 5)

def check_winner(board):
    # Check rows
    for row in board:
        if row[0] == row[1] == row[2] != ' ':
            return True

    # Check columns
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != ' ':
            return True

    # Check diagonals
    if board[0][0] == board[1][1] == board[2][2] != ' ':
        return True
    if board[0][2] == board[1][1] == board[2][0] != ' ':
        return True

    return False

def is_board_full(board):
    for row in board:
        for cell in row:
```

```
            if cell == ' ':
                return False
    return True

def play_game():
    board = [[' ' for _ in range(3)] for _ in range(3)]
    player = 'X'

    while True:
        print_board(board)
        row = int(input("Enter the row (0, 1, or 2): "))
        col = int(input("Enter the column (0, 1, or 2): "))

        if board[row][col] == ' ':
            board[row][col] = player

            if check_winner(board):
                print_board(board)
                print(f"Player {player} wins!")
                break
            elif is_board_full(board):
                print_board(board)
                print("It's a tie!")
                break
            else:
                player = 'O' if player == 'X' else 'X'
        else:
            print("That cell is already occupied. Try again.")

play_game()
```

```
  |   |
-----
  |   |
-----
  |   |
-----
Enter the row (0, 1, or 2): 1
Enter the column (0, 1, or 2): 1
  |   |
-----
  | X |
-----
  |   |
-----
Enter the row (0, 1, or 2): 0
Enter the column (0, 1, or 2): 0
O |   |
```

```
-----
  | X |
-----
  |   |
-----
Enter the row (0, 1, or 2): 0
Enter the column (0, 1, or 2): 1
O | X |
-----
  | X |
-----
  |   |
-----
Enter the row (0, 1, or 2): 2
Enter the column (0, 1, or 2): 1
O | X |
-----
  | X |
-----
  | O |
-----
Enter the row (0, 1, or 2): 2
Enter the column (0, 1, or 2): 0
O | X |
-----
  | X |
-----
X | O |
-----
Enter the row (0, 1, or 2): 2
Enter the column (0, 1, or 2): 2
O | X |
-----
  | X |
-----
X | O | O
-----
Enter the row (0, 1, or 2): 0
Enter the column (0, 1, or 2): 2
O | X | X
-----
  | X |
-----
X | O | O
-----
Player X wins!
```

Interaction Evaluation:

ChatGPT produced a working 2 player tic tac toe game! It doesn't look very good, its interface is awkward, and it's very bare bones, but hey, it works.

4 Prompt 2

Me:

Modify the game so it occurs in a GUI rather than console

ChatGPT:

Sure! Let's modify the game to use the Tkinter library to create a simple graphical user interface (GUI) for the Tic Tac Toe game in Python. Copy and paste this code into a Python script, run it, and you'll get a simple GUI version of Tic Tac Toe. Clicking on any square will place an 'X' or 'O' depending on the turn, and it will check for a winner or a tie after each move.

[2]:
```
import tkinter as tk
from tkinter import messagebox

class TicTacToeGUI:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Tic Tac Toe")
        self.current_player = 'X'
        self.board = [[' ' for _ in range(3)] for _ in range(3)]

        self.buttons = [[None for _ in range(3)] for _ in range(3)]
        for i in range(3):
            for j in range(3):
                self.buttons[i][j] = tk.Button(self.root, text=" ",␣
↪font=('Arial', 20), width=4, height=2,
                                               command=lambda row=i, col=j:␣
↪self.on_button_click(row, col))
                self.buttons[i][j].grid(row=i, column=j)

    def on_button_click(self, row, col):
        if self.board[row][col] == ' ':
            self.board[row][col] = self.current_player
            self.buttons[row][col].config(text=self.current_player)

            if self.check_winner():
                messagebox.showinfo("Winner!", f"Player {self.current_player}␣
↪wins!")
                self.reset_board()
            elif self.is_board_full():
```

```
                messagebox.showinfo("Tie!", "It's a tie!")
                self.reset_board()
            else:
                self.current_player = 'O' if self.current_player == 'X' else 'X'

    def check_winner(self):
        for row in self.board:
            if row[0] == row[1] == row[2] != ' ':
                return True

        for col in range(3):
            if self.board[0][col] == self.board[1][col] == self.board[2][col] !=␣
↪' ':
                return True

        if self.board[0][0] == self.board[1][1] == self.board[2][2] != ' ':
            return True
        if self.board[0][2] == self.board[1][1] == self.board[2][0] != ' ':
            return True

        return False

    def is_board_full(self):
        for row in self.board:
            for cell in row:
                if cell == ' ':
                    return False
        return True

    def reset_board(self):
        self.board = [[' ' for _ in range(3)] for _ in range(3)]
        for i in range(3):
            for j in range(3):
                self.buttons[i][j].config(text=" ")

    def run(self):
        self.root.mainloop()

if __name__ == "__main__":
    game = TicTacToeGUI()
    game.run()
```

```
---------------------------------------------------------------------------
TclError                                  Traceback (most recent call last)
<ipython-input-2-bc460299aef4> in <cell line: 64>()
     63
     64 if __name__ == "__main__":
```

```
---> 65     game = TicTacToeGUI()
     66     game.run()

<ipython-input-2-bc460299aef4> in __init__(self)
      4 class TicTacToeGUI:
      5     def __init__(self):
----> 6         self.root = tk.Tk()
      7         self.root.title("Tic Tac Toe")
      8         self.current_player = 'X'

/usr/lib/python3.10/tkinter/__init__.py in __init__(self, screenName, baseName,␣
 ↪className, useTk, sync, use)
   2297                 baseName = baseName + ext
   2298         interactive = False
-> 2299         self.tk = _tkinter.create(screenName, baseName, className,␣
 ↪interactive, wantobjects, useTk, sync, use)
   2300         if useTk:
   2301             self._loadtk()

TclError: no display name and no $DISPLAY environment variable
```

Interaction Evaluation:

Great .... an error message ...

5 Prompt 3

Me:

TclError Traceback (most recent call last) in <cell line: 64>() 63 64 if **name** == "**main**": —> 65 game = TicTacToeGUI() 66 game.run()

1 frames /usr/lib/python3.10/tkinter/**init**.py in **init**(self, screenName, baseName, className, useTk, sync, use) 2297 baseName = baseName + ext 2298 interactive = False -> 2299 self.tk = _tkinter.create(screenName, baseName, className, interactive, wantobjects, useTk, sync, use) 2300 if useTk: 2301 self._loadtk()

TclError: no display name and no $DISPLAY environment variable

ChatGPT:

It seems you encountered a TclError because you are trying to run the code in an environment where there is no display available. This error commonly occurs in environments like Google Colab, which doesn't support displaying graphical user interfaces directly. To run GUI applications, you typically need to use an environment with a graphical display, such as a local machine or a virtual desktop environment. If you still want to run the code in Google Colab, you can create a simple text-based interface instead of a graphical one, as graphical user interfaces are not supported in

Colab due to its nature as a cloud-based notebook environment. Alternatively, you can run the code locally on your computer to see the graphical interface.

Interaction Evaluation:

ChatGPT seems to have forgotten that we were working in Google Colab once I asked it to create a GUI instead of just a console game. Perhaps, asking for "GUI rather than console" confused it and made it think that we were no longer using Google Colab. Let try again.

6 Prompt 4

Me:

Ok, backtrack to the original tic tac toe game and make it more aesthetic but ensure it still works in google colab

ChatGPT:

Sure! We can create a more aesthetic version of the Tic Tac Toe game in Google Colab by using text-based graphics to represent the board and players' moves. Here's the modified version. This version maintains the functionality of the Tic Tac Toe game while presenting a more aesthetic interface using ASCII art for the board. You can run this code in Google Colab and enjoy playing Tic Tac Toe with a cleaner interface.

[3]:
```
def print_board(board):
    print("-------------")
    for row in board:
        print("|", end="")
        for cell in row:
            print(" " + cell + " |", end="")
        print("\n-------------")

def check_winner(board):
    # Check rows
    for row in board:
        if row[0] == row[1] == row[2] != ' ':
            return True

    # Check columns
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != ' ':
            return True

    # Check diagonals
    if board[0][0] == board[1][1] == board[2][2] != ' ':
        return True
    if board[0][2] == board[1][1] == board[2][0] != ' ':
        return True
```

```
    return False

def is_board_full(board):
    for row in board:
        for cell in row:
            if cell == ' ':
                return False
    return True

def play_game():
    board = [[' ' for _ in range(3)] for _ in range(3)]
    player = 'X'

    while True:
        print_board(board)
        print(f"Player {player}'s turn")
        row = int(input("Enter the row (0, 1, or 2): "))
        col = int(input("Enter the column (0, 1, or 2): "))

        if board[row][col] == ' ':
            board[row][col] = player

            if check_winner(board):
                print_board(board)
                print(f"Player {player} wins!")
                break
            elif is_board_full(board):
                print_board(board)
                print("It's a tie!")
                break
            else:
                player = 'O' if player == 'X' else 'X'
        else:
            print("That cell is already occupied. Try again.")

play_game()
```

```
-------------
|   |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
Player X's turn
Enter the row (0, 1, or 2): 1
Enter the column (0, 1, or 2): 1
```

```
-------------
|   |   |   |
-------------
|   | X |   |
-------------
|   |   |   |
-------------
Player O's turn
Enter the row (0, 1, or 2): 2
Enter the column (0, 1, or 2): 2
-------------
|   |   |   |
-------------
|   | X |   |
-------------
|   |   | O |
-------------
Player X's turn
Enter the row (0, 1, or 2): 0
Enter the column (0, 1, or 2): 1
-------------
|   | X |   |
-------------
|   | X |   |
-------------
|   |   | O |
-------------
Player O's turn
Enter the row (0, 1, or 2): 2
Enter the column (0, 1, or 2): 1
-------------
|   | X |   |
-------------
|   | X |   |
-------------
|   | O | O |
-------------
Player X's turn
Enter the row (0, 1, or 2): 0
Enter the column (0, 1, or 2): 2
-------------
|   | X | X |
-------------
|   | X |   |
-------------
|   | O | O |
-------------
Player O's turn
```

```
Enter the row (0, 1, or 2): 2
Enter the column (0, 1, or 2): 0
-------------
|   | X | X |
-------------
|   | X |   |
-------------
| O | O | O |
-------------
Player O wins!
```

Interaction Evaluation:

Ok, so it definitely looks better. ChatGPT did exactly what I wanted it to! But it is still annoying to input the space where the marker should go.

7 Prompt 5

Me:

Make it so that you can click the space you want to place the marker in

ChatGPT:

To enable clicking on spaces to place markers, we can utilize the ipywidgets library to create a clickable grid. Here's the modified version of Tic Tac Toe game for Google Colab with clickable spaces. This code creates a Tic Tac Toe game where you can click on the spaces to place your marker ('X' or 'O'). It utilizes ipywidgets to create buttons for each space on the board. When a button is clicked, it calls the on_button_click function to update the board and check for a winner. You can run this code in Google Colab and play the game by clicking on the spaces on the board.

[4]:
```python
import ipywidgets as widgets
from IPython.display import display

def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 5)

def check_winner(board):
    # Check rows
    for row in board:
        if row[0] == row[1] == row[2] != ' ':
            return True

    # Check columns
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != ' ':
```

```
            return True

    # Check diagonals
    if board[0][0] == board[1][1] == board[2][2] != ' ':
        return True
    if board[0][2] == board[1][1] == board[2][0] != ' ':
        return True

    return False

def is_board_full(board):
    for row in board:
        for cell in row:
            if cell == ' ':
                return False
    return True

def on_button_click(button, row, col):
    global player, board

    if board[row][col] == ' ':
        button.description = player
        board[row][col] = player

        if check_winner(board):
            print_board(board)
            print(f"Player {player} wins!")
        elif is_board_full(board):
            print_board(board)
            print("It's a tie!")
        else:
            player = 'O' if player == 'X' else 'X'

def create_board_buttons():
    buttons = [[None for _ in range(3)] for _ in range(3)]
    for i in range(3):
        for j in range(3):
            button = widgets.Button(description=" ", layout=widgets.
 ↪Layout(width='50px', height='50px'))
            button.on_click(lambda _, row=i, col=j: on_button_click(button, row,␣
 ↪col))
            buttons[i][j] = button
    return buttons

def play_game():
    global player, board
    player = 'X'
```

```
    board = [[' ' for _ in range(3)] for _ in range(3)]
    print_board(board)
    display(widgets.GridBox(create_board_buttons(), layout=widgets.
 ↪Layout(grid_template_columns="repeat(3, 50px)")))

play_game()
```

```
  |   |
-----
  |   |
-----
  |   |
-----
```

```
---------------------------------------------------------------------------
TraitError                                 Traceback (most recent call last)
<ipython-input-4-a368f4d41075> in <cell line: 67>()
     65     display(widgets.GridBox(create_board_buttons(), layout=widgets.
 ↪Layout(grid_template_columns="repeat(3, 50px)")))
     66
---> 67 play_game()

<ipython-input-4-a368f4d41075> in play_game()
     63     board = [[' ' for _ in range(3)] for _ in range(3)]
     64     print_board(board)
---> 65     display(widgets.GridBox(create_board_buttons(), layout=widgets.
 ↪Layout(grid_template_columns="repeat(3, 50px)")))
     66
     67 play_game()

/usr/local/lib/python3.10/dist-packages/ipywidgets/widgets/widget_box.py in␣
 ↪__init__(self, children, **kwargs)
     62     def __init__(self, children=(), **kwargs):
     63         kwargs['children'] = children
---> 64         super(Box, self).__init__(**kwargs)
     65         self.on_displayed(Box._fire_children_displayed)
     66

/usr/local/lib/python3.10/dist-packages/ipywidgets/widgets/widget.py in␣
 ↪__init__(self, **kwargs)
    475         """Public constructor"""
    476         self._model_id = kwargs.pop('model_id', None)
--> 477         super(Widget, self).__init__(**kwargs)
    478
    479         Widget._call_widget_constructed(self)

/usr/local/lib/python3.10/dist-packages/traitlets/traitlets.py in __init__(self,␣
 ↪*args, **kwargs)
```

```
   1341             for key, value in kwargs.items():
   1342                 if self.has_trait(key):
-> 1343                     setattr(self, key, value)
   1344                     changes[key] = Bunch(
   1345                         name=key,

/usr/local/lib/python3.10/dist-packages/traitlets/traitlets.py in __set__(self,␣
↪obj, value)
    727             raise TraitError('The "%s" trait is read-only.' % self.name)
    728         else:
--> 729             self.set(obj, value)
    730
    731     def _validate(self, obj, value):

/usr/local/lib/python3.10/dist-packages/traitlets/traitlets.py in set(self, obj,␣
↪value)
    701
    702     def set(self, obj, value):
--> 703         new_value = self._validate(obj, value)
    704         try:
    705             old_value = obj._trait_values[self.name]

/usr/local/lib/python3.10/dist-packages/traitlets/traitlets.py in _validate(self,␣
↪obj, value)
    733             return value
    734         if hasattr(self, "validate"):
--> 735             value = self.validate(obj, value)
    736         if obj._cross_validation_lock is False:
    737             value = self._cross_validate(obj, value)

/usr/local/lib/python3.10/dist-packages/traitlets/traitlets.py in validate(self,␣
↪obj, value)
   2854             return value
   2855
-> 2856         value = self.validate_elements(obj, value)
   2857
   2858         return value

/usr/local/lib/python3.10/dist-packages/traitlets/traitlets.py in␣
↪validate_elements(self, obj, value)
   2866                 v = self._trait._validate(obj, v)
   2867             except TraitError as error:
-> 2868                 self.error(obj, v, error)
   2869             else:
   2870                 validated.append(v)

/usr/local/lib/python3.10/dist-packages/traitlets/traitlets.py in error(self, obj,␣
↪value, error, info)
```

```
    818                         ),
    819                     )
--> 820             raise error
    821         else:
    822             # this trait caused an error

/usr/local/lib/python3.10/dist-packages/traitlets/traitlets.py in␣
↪validate_elements(self, obj, value)
   2864         for v in value:
   2865             try:
-> 2866                 v = self._trait._validate(obj, v)
   2867             except TraitError as error:
   2868                 self.error(obj, v, error)

/usr/local/lib/python3.10/dist-packages/traitlets/traitlets.py in _validate(self,␣
↪obj, value)
    733             return value
    734         if hasattr(self, "validate"):
--> 735             value = self.validate(obj, value)
    736         if obj._cross_validation_lock is False:
    737             value = self._cross_validate(obj, value)

/usr/local/lib/python3.10/dist-packages/traitlets/traitlets.py in validate(self,␣
↪obj, value)
   2145             return value
   2146         else:
-> 2147             self.error(obj, value)
   2148
   2149     def info(self):

/usr/local/lib/python3.10/dist-packages/traitlets/traitlets.py in error(self, obj␣
↪value, error, info)
    823             if self.name is None:
    824                 # this is not the root trait
--> 825                 raise TraitError(value, info or self.info(), self)
    826             else:
    827                 # this is the root trait

TraitError: The 'children' trait of a GridBox instance contains an Instance of a␣
↪TypedTuple which expected a Widget, not the list [Button(description=' ',␣
↪layout=Layout(height='50px', width='50px'), style=ButtonStyle()),␣
↪Button(description=' ', layout=Layout(height='50px', width='50px'),␣
↪style=ButtonStyle()), Button(description=' ', layout=Layout(height='50px',␣
↪width='50px'), style=ButtonStyle())].
```

Interaction Evaluation:

. . . error #2. . .

8 Prompt 6

Me:

TraitError Traceback (most recent call last) in <cell line: 67>() 65 display(widgets.GridBox(create_board_buttons(), layout=widgets.Layout(grid_template_columns="repeat(3, 50px)"))) 66 —> 67 play_game()

13 frames /usr/local/lib/python3.10/dist-packages/traitlets/traitlets.py in error(self, obj, value, error, info) 823 if self.name is None: 824 # this is not the root trait –> 825 raise TraitError(value, info or self.info(), self) 826 else: 827 # this is the root trait

TraitError: The 'children' trait of a GridBox instance contains an Instance of a TypedTuple which expected a Widget, not the list [Button(description=' ', layout=Layout(height='50px', width='50px'), style=ButtonStyle()), Button(description=' ', layout=Layout(height='50px', width='50px'), style=ButtonStyle()), Button(description=' ', layout=Layout(height='50px', width='50px'), style=ButtonStyle())].

ChatGPT:

Apologies for the confusion. It seems there was an issue with creating the grid of buttons using GridBox. Let's adjust the code to use HBox and VBox instead. Here's the modified version This version creates a Tic Tac Toe game where you can click on the spaces to place your marker ('X' or 'O'). It utilizes ipywidgets to create buttons for each space on the board. When a button is clicked, it calls the on_button_click function to update the board and check for a winner. You can run this code in Google Colab and play the game by clicking on the spaces on the board.

[5]:
```
import ipywidgets as widgets
from IPython.display import display

def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 5)

def check_winner(board):
    # Check rows
    for row in board:
        if row[0] == row[1] == row[2] != ' ':
            return True

    # Check columns
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != ' ':
            return True

    # Check diagonals
    if board[0][0] == board[1][1] == board[2][2] != ' ':
        return True
```

```
    if board[0][2] == board[1][1] == board[2][0] != ' ':
        return True

    return False

def is_board_full(board):
    for row in board:
        for cell in row:
            if cell == ' ':
                return False
    return True

def on_button_click(button, row, col):
    global player, board

    if board[row][col] == ' ':
        button.description = player
        board[row][col] = player

        if check_winner(board):
            print_board(board)
            print(f"Player {player} wins!")
        elif is_board_full(board):
            print_board(board)
            print("It's a tie!")
        else:
            player = 'O' if player == 'X' else 'X'

def create_board_buttons():
    buttons = [[None for _ in range(3)] for _ in range(3)]
    for i in range(3):
        for j in range(3):
            button = widgets.Button(description=" ", layout=widgets.
 ↪Layout(width='50px', height='50px'))
            button.on_click(lambda _, row=i, col=j: on_button_click(button, row,␣
 ↪col))
            buttons[i][j] = button
    return buttons

def play_game():
    global player, board
    player = 'X'
    board = [[' ' for _ in range(3)] for _ in range(3)]
    print_board(board)
    display(widgets.VBox([widgets.HBox(create_board_buttons()[i]) for i in␣
 ↪range(3)]))
```

```
play_game()
```

```
  |   |
-----
  |   |
-----
  |   |
-----
```

```
VBox(children=(HBox(children=(Button(description=' ', layout=Layout(height='50px',␣
 ↪width='50px'), style=Button...
```

```
X |   | O
-----
X | O |
-----
X | O | X
-----
Player X wins!
```

Interaction Evaluation:

Now I can click to place markers, but I can't see intermediate boards. I have to play the game "blind" and it only outputs the result once someone wins.

9 Prompt 7

Me:

Now I can click to place markers, but I cannot see intermediate boards and it is not very aesthetic

ChatGPT:

Apologies for the oversight. Let's modify the code to display the current state of the board after each move and make it more aesthetic by using a grid layout with better spacing. We'll also display a message indicating the current player's turn and the result of the game (win or tie). Here's the updated version. This version now displays the current state of the board after each move and shows a message indicating the current player's turn. Additionally, it checks for a winner or a tie after each move and stops the game accordingly. You can run this code in Google Colab, and it will display a more aesthetic and interactive Tic Tac Toe game.

[6]:
```
import ipywidgets as widgets
from IPython.display import display, clear_output

def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 5)
```

```
def check_winner(board):
    # Check rows
    for row in board:
        if row[0] == row[1] == row[2] != ' ':
            return True

    # Check columns
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != ' ':
            return True

    # Check diagonals
    if board[0][0] == board[1][1] == board[2][2] != ' ':
        return True
    if board[0][2] == board[1][1] == board[2][0] != ' ':
        return True

    return False

def is_board_full(board):
    for row in board:
        for cell in row:
            if cell == ' ':
                return False
    return True

def on_button_click(button, row, col):
    global player, board

    if board[row][col] == ' ':
        button.description = player
        board[row][col] = player

        if check_winner(board):
            print_board(board)
            print(f"Player {player} wins!")
            return True
        elif is_board_full(board):
            print_board(board)
            print("It's a tie!")
            return True
        else:
            player = 'O' if player == 'X' else 'X'
            return False

def create_board_buttons():
    buttons = [[None for _ in range(3)] for _ in range(3)]
```

```
    for i in range(3):
        for j in range(3):
            button = widgets.Button(description=" ", layout=widgets.
 ↪Layout(width='50px', height='50px'))
            button.on_click(lambda _, row=i, col=j: on_button_click(button, row,␣
 ↪col))
            buttons[i][j] = button
    return buttons

def display_board(board):
    clear_output()
    print_board(board)
    display(widgets.VBox([widgets.HBox(create_board_buttons()[i]) for i in␣
 ↪range(3)]))

def play_game():
    global player, board
    player = 'X'
    board = [[' ' for _ in range(3)] for _ in range(3)]

    display_board(board)
    print(f"Player {player}'s turn")

    while True:
        if player == 'X':
            print(f"Player {player}'s turn")
        else:
            print(f"Player {player}'s turn")

        if check_winner(board) or is_board_full(board):
            break
        else:
            pass

play_game()
```

```
  |   |
-----
  |   |
-----
  |   |
-----
```

```
VBox(children=(HBox(children=(Button(description=' ', layout=Layout(height='50px',␣
 ↪width='50px'), style=Button...
```

```
Streaming output truncated to the last 5000 lines.
Player X's turn
```

```
Player X's turn
Player X's turn
Player X's turn
Player X's turn
Player X's turn
Player X's turn
Player X's turn
Player X's turn
Player X's turn
Player X's turn
...
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-6-c2752bf24024> in <cell line: 87>()
     85             pass
     86
---> 87 play_game()

<ipython-input-6-c2752bf24024> in play_game()
     76     while True:
     77         if player == 'X':
---> 78             print(f"Player {player}'s turn")
     79         else:
     80             print(f"Player {player}'s turn")

/usr/local/lib/python3.10/dist-packages/ipykernel/iostream.py in write(self,␣
 ↪string)
    400             is_child = (not self._is_master_process())
    401             # only touch the buffer in the IO thread to avoid races
--> 402             self.pub_thread.schedule(lambda : self._buffer.write(string))
    403             if is_child:
    404                 # mp.Pool cannot be trusted to flush promptly (or ever),

/usr/local/lib/python3.10/dist-packages/ipykernel/iostream.py in schedule(self, f
    201             self._events.append(f)
    202             # wake event thread (message content is ignored)
--> 203             self._event_pipe.send(b'')
    204         else:
    205             f()

/usr/local/lib/python3.10/dist-packages/zmq/sugar/socket.py in send(self, data,␣
 ↪flags, copy, track, routing_id, group)
    616                 )
    617             data.group = group
--> 618         return super().send(data, flags=flags, copy=copy, track=track)
    619
```

```
    620     def send_multipart(

zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.send()

zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.send()

zmq/backend/cython/socket.pyx in zmq.backend.cython.socket._send_copy()

/usr/local/lib/python3.10/dist-packages/zmq/backend/cython/checkrc.pxd in zmq.
↪backend.cython.checkrc._check_rc()

KeyboardInterrupt:
```

Interaction Evaluation:

Now there's an infinite loop....

10 Prompt 8

Me:

Now the console keeps displaying "X's turn" before I can make a move

ChatGPT:

Apologies for the confusion. Let's adjust the code so that the message indicating the current player's turn only appears after a move has been made. Here's the updated version. This adjustment ensures that the message indicating the current player's turn only appears after a move has been made. You can run this code in Google Colab, and it will display the Tic Tac Toe game with the correct sequence of messages.

[8]:
```
import ipywidgets as widgets
from IPython.display import display, clear_output

def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 5)

def check_winner(board):
    # Check rows
    for row in board:
        if row[0] == row[1] == row[2] != ' ':
            return True

    # Check columns
    for col in range(3):
```

```
        if board[0][col] == board[1][col] == board[2][col] != ' ':
            return True

    # Check diagonals
    if board[0][0] == board[1][1] == board[2][2] != ' ':
        return True
    if board[0][2] == board[1][1] == board[2][0] != ' ':
        return True

    return False

def is_board_full(board):
    for row in board:
        for cell in row:
            if cell == ' ':
                return False
    return True

def on_button_click(button, row, col):
    global player, board

    if board[row][col] == ' ':
        button.description = player
        board[row][col] = player

        if check_winner(board):
            print_board(board)
            print(f"Player {player} wins!")
            return True
        elif is_board_full(board):
            print_board(board)
            print("It's a tie!")
            return True
        else:
            player = 'O' if player == 'X' else 'X'
            display_board(board)
            print(f"Player {player}'s turn")
            return False

def create_board_buttons():
    buttons = [[None for _ in range(3)] for _ in range(3)]
    for i in range(3):
        for j in range(3):
            button = widgets.Button(description=" ", layout=widgets.
 ↪Layout(width='50px', height='50px'))
            button.on_click(lambda _, row=i, col=j: on_button_click(button, row,␣
 ↪col))
```

```
            buttons[i][j] = button
    return buttons

def display_board(board):
    clear_output()
    print_board(board)
    display(widgets.VBox([widgets.HBox(create_board_buttons()[i]) for i in␣
 ↪range(3)]))

def play_game():
    global player, board
    player = 'X'
    board = [[' ' for _ in range(3)] for _ in range(3)]

    display_board(board)
    print(f"Player {player}'s turn")

    while True:
        if check_winner(board) or is_board_full(board):
            break
        else:
            pass

play_game()
```

```
X | X |
-----
  |   |
-----
O | O |
-----
```

```
VBox(children=(HBox(children=(Button(description=' ', layout=Layout(height='50px',␣
 ↪width='50px'), style=Button...
```

```
Player X's turn
X | X | X
-----
  |   |
-----
O | O |
-----
Player X wins!
```

Interaction Evaluation:

Now it shows some of the intermediate boards, but only upon execution interruption. Otherwise, it's exactly as it was before.

11 Prompt 9

Me:

Now the code does not show any output until I interrupt its execution

ChatGPT:

My apologies for the oversight. It seems that the loop in the play_game() function is preventing the code from displaying any output until it exits. To address this issue, we need to continuously check for the game's status and update the display accordingly. We can achieve this by removing the loop and adding the necessary logic to the on_button_click() function. Here's the updated code. With this modification, the game will display the current state of the board and the appropriate message after each move. You can run this code in Google Colab, and it will provide interactive gameplay with updated output after each move.

[9]:
```
import ipywidgets as widgets
from IPython.display import display, clear_output

def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 5)

def check_winner(board):
    # Check rows
    for row in board:
        if row[0] == row[1] == row[2] != ' ':
            return True

    # Check columns
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != ' ':
            return True

    # Check diagonals
    if board[0][0] == board[1][1] == board[2][2] != ' ':
        return True
    if board[0][2] == board[1][1] == board[2][0] != ' ':
        return True

    return False

def is_board_full(board):
    for row in board:
        for cell in row:
            if cell == ' ':
                return False
```

```
    return True

def on_button_click(button, row, col):
    global player, board

    if board[row][col] == ' ':
        button.description = player
        board[row][col] = player

        if check_winner(board):
            print_board(board)
            print(f"Player {player} wins!")
        elif is_board_full(board):
            print_board(board)
            print("It's a tie!")
        else:
            player = 'O' if player == 'X' else 'X'
            print_board(board)
            print(f"Player {player}'s turn")

def create_board_buttons():
    buttons = [[None for _ in range(3)] for _ in range(3)]
    for i in range(3):
        for j in range(3):
            button = widgets.Button(description=" ", layout=widgets.
 ↪Layout(width='50px', height='50px'))
            button.on_click(lambda _, row=i, col=j: on_button_click(button, row,␣
 ↪col))
            buttons[i][j] = button
    return buttons

def display_board(board):
    clear_output()
    print_board(board)
    display(widgets.VBox([widgets.HBox(create_board_buttons()[i]) for i in␣
 ↪range(3)]))

def play_game():
    global player, board
    player = 'X'
    board = [[' ' for _ in range(3)] for _ in range(3)]

    display_board(board)
    print(f"Player {player}'s turn")

play_game()
```

```
  |   |
-----
  |   |
-----
  |   |
-----
```

```
VBox(children=(HBox(children=(Button(description=' ', layout=Layout(height='50px',␣
 ↪width='50px'), style=Button...
```

```
Player X's turn
X |   |
-----
  |   |
-----
  |   |
-----
Player O's turn
X | O |
-----
  |   |
-----
  |   |
-----
Player X's turn
X | O |
-----
  |   |
-----
X |   |
-----
Player O's turn
X | O |
-----
  |   |
-----
X |   | O
-----
Player X's turn
X | O |
-----
X |   |
-----
X |   | O
-----
Player X wins!
```

Interaction Evaluation:

Ok, were getting somewhere again! The controller at the top works to place the markers and intermediate boards are shown. However, we can keep placing markers once someone wins and it's not very aesthetic.

12 Prompt 10

Me:

Can you make it more aesthetic?

ChatGPT:

Certainly! Let's enhance the aesthetics of the game by using Unicode characters for the board and markers. Here's the updated code. Now the game features an aesthetically pleasing board with Unicode characters. You can run this code in Google Colab, and it will provide an improved visual experience for playing Tic Tac Toe.

[10]:
```
import ipywidgets as widgets
from IPython.display import display, clear_output

def print_board(board):
    print("")
    for row in board:
        print(" " + "  ".join(row) + " ")
        print("")
    print("")

def check_winner(board):
    # Check rows
    for row in board:
        if row[0] == row[1] == row[2] != ' ':
            return True

    # Check columns
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != ' ':
            return True

    # Check diagonals
    if board[0][0] == board[1][1] == board[2][2] != ' ':
        return True
    if board[0][2] == board[1][1] == board[2][0] != ' ':
        return True

    return False

def is_board_full(board):
```

```
    for row in board:
        for cell in row:
            if cell == ' ':
                return False
    return True

def on_button_click(button, row, col):
    global player, board

    if board[row][col] == ' ':
        button.description = player
        board[row][col] = player

        if check_winner(board):
            print_board(board)
            print(f"Player {player} wins!")
        elif is_board_full(board):
            print_board(board)
            print("It's a tie!")
        else:
            player = 'O' if player == 'X' else 'X'
            print_board(board)
            print(f"Player {player}'s turn")

def create_board_buttons():
    buttons = [[None for _ in range(3)] for _ in range(3)]
    for i in range(3):
        for j in range(3):
            button = widgets.Button(description=" ", layout=widgets.
 ↪Layout(width='50px', height='50px'))
            button.on_click(lambda _, row=i, col=j: on_button_click(button, row,␣
 ↪col))
            buttons[i][j] = button
    return buttons

def display_board(board):
    clear_output()
    print_board(board)
    display(widgets.VBox([widgets.HBox(create_board_buttons()[i]) for i in␣
 ↪range(3)]))

def play_game():
    global player, board
    player = 'X'
    board = [[' ' for _ in range(3)] for _ in range(3)]

    display_board(board)
```

```
    print(f"Player {player}'s turn")

play_game()
```

```
-------------
|   |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
-------------
```

```
VBox(children=(HBox(children=(Button(description=' ', layout=Layout(height='50px',␣
 ↪width='50px'), style=Button...
```

```
Player X's turn
-------------
| X |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
-------------
Player O's turn
-------------
| X |   |   |
-------------
|   | O |   |
-------------
|   |   |   |
-------------
-------------
Player X's turn
-------------
| X |   |   |
-------------
|   | O |   |
-------------
|   |   | X |
-------------
-------------
Player O's turn
-------------
| X |   | O |
-------------
|   | O |   |
```

```
-------------
|   |   | X |
-------------
-------------
Player X's turn
-------------
| X |   | O |
-------------
|   | O |   |
-------------
| X |   | X |
-------------
-------------
Player O's turn
-------------
| X |   | O |
-------------
|   | O |   |
-------------
| X | O | X |
-------------
-------------
Player X's turn
-------------
| X |   | O |
-------------
| X | O |   |
-------------
| X | O | X |
-------------
-------------
Player X wins!
```

Interaction Evaluation:

It looks better now, but for some reason there's an extra row. I would keep going, but as per the premise, I am impatient, so this will have to do for the final product.

13 Advantages and Limitations of ChatGPT

For this specific task, ChatGPT was successful in completing simple tasks that were console output based. It successfully created the basic logic and console output for the game from the initial prompt (prompt 1). Additionally, it was able to successfully create a more aesthetic console output (prompt 4). ChatGPT also had limited success in fixing code according to the error message produced in the output (prompts 3 and 6). ChatGPT was able to explain why the error occurred, and how it attempted to fix it. For prompt 3, this was especially useful, as ChatGPT revealed that it forgot the context that it was operating in (Google Colab, not a local python IDE).

Two specific limitations of ChatGPT that appeared during the test was that it seemed to forget the

context that it is operating in (prompt 2) and produced erroneous code when prompted to include more advanced features such as GUIs and more intricate game controllers (prompts 2, 5, 7). This seems to imply a limit on the complexity of the operations and tasks that ChatGPT can complete. Creating a basic command line tic tac toe game is elementary, advancing that same idea with a GUI, controller, AI, and other features, is much more advanced. The number of online examples of the basic tic tac toe games is likely innumerable. In comparison, source code for advanced implementations of tic tac toe are likely seldom in comparison. This is especially true when you consider the constraints that we gave ChatGPT: the tic tac toe game must operate in Google Colab. This adds another layer of complexity that further diminishes the performance of ChatGPT.

Despite the former, within 10 iterations, ChatGPT was able to produce a tic tac toe game that properly functions and accepts input through a click-based game controller. Compared to a human programmer, ChatGPT was able to accomplish this at a rapid pace. This suggests that ChatGPT based programming could prove useful for tasks such as rapid experimentation, where getting messy results is more important that thorough planning. Furthermore, I was unaware of the available libraries that I could use to create such a controller. By generating partially erroneous code, ChatGPT exposed me to ipywidgets, which I could explore, leading to further solutions indirectly stemming from ChatGPT. ChatGPT seems to be better suited to idea generation, inspiration, and rapid (albeit messy and erroneous) development for the sake of exploration. It is not a perfect system, does not handle complexity well, and should not be used to create code that is expected to be anywhere near production quality.

14 NLP behind ChatGPT

ChatGPT, which is based on the GPT 3.5 model, uses a transformer at its core [5]. A transformer is a type of neural network that uses attention to perform well at a variety of tasks, such as natural language processing. Creating large models using transformers leads to neural networks called large language models, or LLMS. These models have billions of trainable parameters and use huge corpuses of text to train. In the case of ChatGPT, it was trained on a massive corpus of text, including books, articles, and websites, to predict the next word in a sequence of text, given the prior words. Additionally, ChatGPT was fine-tuned using supervised learning and optimized using the PPO reinforcement learning algorithm. This fine-tuning process allows ChatGPT to generalize to a wide variety of tasks, such as text generation, classification, sentiment analysis, and more.

Essentially, this means that ChatGPT is only as good as what it was trained on. At its core, ChatGPT is a neural network that was trained on a very large dataset. Neural networks can only model data accurately if it is within the scope of the training data. If it is outside, a model may lose accuracy as it was not intended to handle that data. In practice, this means that as long as we ask ChatGPT to do something that was within its training cycle, it will have a reasonable chance of performing successfully. The more common the task is within its training, the better the outcome. This means that complex tasks that combine many niche ideas and tasks will likely be poorly handled by ChatGPT. This perfectly explains ChatGPT's relative failure to create a GUI-like interface for the tic tac toe game on Google Colab. The situation is likely outside of its training domain. Furthermore, its limited error fixing capability stems from the fact that error messages are often unique to the situation, despite similarities across many common error types. On the other hand, creating the basic tic tac toe game is a common task that is likely found many times in its training corpus. As such, ChatGPT excels at this. A similar situation is when ChatGPT excelled at creating ASCII art (prompt 4). Finally, LLMs have been known to hallucinate [8] and forget

their operating context, which explains why ChatGPT forgot that we were using Google Colab, and perhaps why it generated erroneous code.

To improve the ability of ChatGPT to complete programming related tasks, I would fine-tune the model to cater to such a context. OpenAI made this possible through GPT 3.5 fine tuning. Thus, my suggestion for improving the efficacy of ChatGPT in this specific scenario is twofold: fine-tune the model, and carefully evaluate the results using a framework (such as described in [6]).

15 Discussions and Conclusions

Overall, ChatGPT appears to be useful for quickly generating code for monotonous, simple tasks. In this specific instance, ChatGPT would have been very helpful to create the basic tic tac toe game logic and console output. We could have then manually programmed to create more advanced features such as a better UI, game controller, and perhaps an AI opponent. I have often used ChatGPT in my personal projects to retrieve generic code for common use cases of certain python libraries. For instance, I ask ChatGPT to generate a scatter plot using matplotlib. It gives me easy to change skeleton code, that I can then adapt to my data. If I then want to change the style, I can easily ask ChatGPT to change the color, markers, etc. These are all easily accomplished tasks that can be found in the library's documentation. However, ChatGPT fails to go beyond this and connect/combine complex ideas and libraries. This was demonstrated in ChatGPT's relative failure to create a better UI and controller for the tic tac toe game. Thus, in the scope of programming, it seems that ChatGPT is most useful as a tool to save developers time by creating code for generic, basic use cases (see [7] for related ideas).

Furthermore, ChatGPT is a valuable tool for exploration. Asking it to complete complex tasks will likely lead to failure, but a productive type of failure. The incorrect code it provides may expose you to new libraries and ideas. For instance, through the erroneous code produced at prompt 5, I learned about the ipywidgets library. I could have then explored the documentation for the library and created my own controller without ChatGPT. However, ChatGPT provided the indirect help of showing me a potentially helpful library. In short, ChatGPT is useful for rapid experimentation without expecting any guarantee of correctness (see [6] for evaluation of LLMs for code generation).

16 References

1. https://chat.openai.com/
2. https://colab.research.google.com/
3. https://openai.com/
4. https://www.python.org/psf-landing/
5. https://www.sciencedirect.com/science/article/pii/S266734522300024X?ref=pdf_download&fr=RR-9&rr=863639888da4db1d
6. https://proceedings.neurips.cc/paper_files/paper/2023/file/43e9d647ccd3e4b7b5baab53f0368686-Paper-Conference.pdf
7. https://mesopotamian.press/journals/index.php/cs/article/view/51/71

8. https://machinelearningmastery.com/a-gentle-introduction-to-hallucinations-in-large-language-models/