A Case Study for using Generative Language Models in GUI Development

Katelyn Reagan
Smith College
kreagan@smith.edu

Maryam Berijanian
Michigan State University
berijani@msu.edu

Dirk Colbry
Michigan State University
colbrydi@msu.edu

ABSTRACT

In the age of advanced open-source artificial intelligence (AI) and a growing demand for software tools, programming skills are as important as ever. For even the most experienced programmers, it can be challenging to determine which software libraries and packages are best suited to fit specific programming needs. To investigate the potential of AI-supported learning, this case study explores the use of OpenAI's ChatGPT, powered by GPT-3.5 and GPT-4, by students to create an image annotation graphical user interface (GUI) in Python. This task was selected because good User Experience (UX) design is a deceptively complex task in that it can be very easy to build a GUI interface but extremely hard to build one that is well designed. The approaches employed in this study included creating a program from scratch that integrates the listed features incrementally; compiling a list of essential features and requesting ChatGPT to modify existing code accordingly; collaborating on specific segments of a user-initiated program; and creating a program anew using GPT-4 for comparative analysis. The findings of this case study indicate that ChatGPT is optimally utilized for responding to precise queries rather than creating code from scratch. Effective use of ChatGPT requires a foundational understanding of programming languages. As a learning tool, ChatGPT can help a novice programmer create competent initial drafts, akin to what one might expect from an introductory programming course, yet they necessitate substantial modifications for deployment of the tool even as a prototype.

KEYWORDS

Generative Language Models, ChatGPT, Graphical User Interface (GUI), User Experience (UX) Design, Software Tools, Artificial Intelligence in Education (AIED)

1 BACKGROUND

ChatGPT is a web-based chat-bot created by OpenAI [9] and launched in November 2022. After reaching over one million users within five days of launching [2], the chat-bot has continued to gain popularity while demonstrating its ability to convey its broad knowledge using natural language. A Generative Pre-trained Transformer (GPT) powers the chat-bot, enabling original text generation that mimics the language model's training data [3]. The primary iteration of ChatGPT employed in this study was GPT-3.5, accessible freely through an OpenAI account. GPT-4, requiring a premium account for access, was then utilized for comparative analysis with GPT-3.5. This study's emphasis on GPT-3.5 aims to illustrate the use of a technology that is readily available to students. Currently, ChatGPT is not without issues. The program is prone to exuding confidence while formulating statements from flawed logic and providing compelling disinformation [7]. Yet, since this particular chat-bot is still relatively new and continues to get updated and supported by OpenAI [1], the technology has a promising future.

Many software tools and libraries exist to assist in the creation of a graphical user interface (GUI), but finding a specific tool that fits every need of programming task can be time-consuming. Additionally, professional, public facing interfaces must meet the diversity of users' needs and put accessibility at the forefront, ensuring intuitive functionality and smooth navigation of the tools. The design of "production" quality GUIs calls for unwavering attention to detail and likely requires expertise to create functional layouts [8].

According to the chat-bot itself, ChatGPT trained on "a diverse range of internet text up until the knowledge cutoff date in September 2021". Additionally, it can instruct beginner programmers on GUI creation through explanations of concepts, code examples, event handling, problem-solving, and learning material recommendations. The promise of such capacities could help even inexperienced coders expedite the creation of custom GUIs.

2 CASE STUDY

Although deceivingly simple, the creation of a visually intuitive and smooth-functioning GUI requires a moderate degree of interface programming expertise. A series of back-and-forth interactions with ChatGPT demonstrate its ability to guide programmers without user interface programming experience and minimal knowledge beyond a basic definition of a GUI. This challenging project was specifically selected to see how far ChatGPT could be pushed as a learning tool. The particular program was intended to create a segmentation tool to annotate images [5]. Image segmentation is the process wherein the user uses pixel selection tools to identify regions of an image that are of interest (foreground). Image annotation is often the first step in many machine learning and scientific image understanding workflows.

The desired segmentation program must enable the user to easily upload an image, segment the image, and save the segmented image. The user must be able to switch between annotation colors and zoom/pan the image. The resulting segmented labeling of the output must maintain a size identical to that of the original so that coordinate points for the labels correlate directly with the pixels in

https://doi.org/10.22369/issn.2153-4136/15/2/6

the image. Other desirable features include undo/redo buttons, the option to select different colors, a fill bucket or lasso fill tool, and a size scale to change the size of the markups. Ideally, the program would be written in Python and exclude any excess features that would slow the segmentation process down or provide undesirable results.

Several techniques were used to prompt the chat-bot, including:

Method 1 – From Scratch Evolution: asks ChatGPT to create a Python-based image annotation code from scratch, gradually introducing new features into the program.
Method 2 – From Sample: gives ChatGPT code for a previously created GUI that functions but does not have all required features, gradually fixing and developing the code as needed.
Method 3 – From Scratch List Requirements: gives ChatGPT an extensive list of requested features to create a GUI from scratch.
Method 4 – From Scratch Assisted: gives ChatGPT a rough baseline code inspired by previously established code and knowledge gained through the other methods' results.
Method 5 – Using GPT-4: employs GPT-4 to compare the results with those of GPT-3.5.

Each technique was attempted anywhere from one to three times, depending on whether the programmers felt another attempt would be constructive in demonstrating the capability of the chat-bot given different initial conditions.

A complete list of the transcripts used throughout this paper can be found on the following openly available repository: https://github.com/katiereagan/ChatGPT-Conversation.

2.1 Method 1: From Scratch Evolution

The method "From Scratch Evolution" starts by asking ChatGPT to create a Python-based image annotation code from scratch. Using this code as a baseline, new features were gradually introduced into the program.

Attempt 1:
The initial prompt was *"Python-based image annotation program"*. The output (Figure 1) consisted of a basic annotation program using Open Source Computer Vision Library (OpenCV) [10], with the steps outlined above each piece of code.

Underneath the code, ChatGPT provides a brief description that walks the user through each step of the program, noting that OpenCV must be installed in the Python environment:

ChatGPT:
In this code, the image is loaded using `cv2.imread()` *and displayed in a window using* `cv2.imshow()`. *The user can click on the image, and the* `annotate()` *function is called to handle the mouse events. When a point is clicked, it is added to the points list, and the user is prompted to enter an annotation. The point and annotation are stored in separate lists. The annotated points are drawn on the image, and the updated image is displayed using* `cv2.imshow()`. *After the user is done annotating, the annotated points and annotations are printed, and the annotated image is saved using* `cv2.imwrite()`. *Finally, the image window is closed using* `cv2.destroyAllWindows()`.

Note that this is a basic example, and you may need to customize it based on your specific requirements. Additionally, you'll need to have OpenCV installed in your Python environment to run this code.

Upon further prompting, ChatGPT provided the user with more detailed instructions about installations of OpenCV using the terminal. This initial program did not function properly, generating errors with each run that were outside the programmer's ability to fix. Rather than searching the code for errors, ChatGPT was prompted to create a program that did not use OpenCV. The alternative offered (Figure 2) by the chat-bot depended on Pillow (Python Imaging Library or PIL fork) [6] for image manipulation and Tkinter (Tk interface) [13] for the user interface. The steps outlined in the code and description provided were similar to those used in the OpenCV-based program, varying in the libraries used.

Again, the program generated was a Python app that would not respond, attempting to but never successfully opening a window. This issue could be due to outdated suggestions due to ChatGPT's information intake cutoff occurring in September of 2021, nearly two years prior to this study.

When working with a chat-bot, the general process is to suggest a new feature, debug, and repeat while gradually introducing new features into the program or correcting pre-existing features.

One notable benefit of working with ChatGPT is that, after getting output, the user can ask the system questions about its code to get more insight into the logic. The chat-bot's conversational nature benefits beginners, who may be less familiar with user interface design.

Although ChatGPT has a reputation for producing solutions that are not necessarily accurate, this work found that ChatGPT's explanations could at least serve as a starting point for further explanation.

Ultimately, ChatGPT did not successfully produce a working code and it was decided to start a second attempt guided by the knowledge and experience gained from the first attempt.

Attempt 2:
For the second attempt, the order in which the features were requested changed in hopes of producing a different outcome.

The initial prompt was still *"Python-based image annotation code"*. The code generated once again depended on OpenCV, so a request was made to rewrite the program without using OpenCV. The default alternative seems to be PIL. To better fit the program requirements, the prompt requested the system record the coordinate points of the user annotations. The resulting code (Figure 3) added a portion of code dedicated to this new feature.

The system struggled to open the Python application popup. After several rounds of requests to optimize the provided code to facilitate the generation of the app window, the program remained non-functional. The chat-bot cited techniques such as double buffering, resizing the image, and using batch annotations, but the final product remained unable to load. It became increasingly apparent that ChatGPT struggles to create its code from scratch but can explain pre-existing code. ChatGPT would benefit from more user

```python
import cv2

# Load the image
image = cv2.imread('image.jpg')

# Create a window and bind it to the image
cv2.namedWindow('Image')
cv2.imshow('Image', image)

# Create lists to store annotation coordinates
points = []
annotations = []

# Callback function for mouse events
def annotate(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        # Add the clicked point to the list
        points.append((x, y))
        # Prompt the user to input annotation
        annotation = input('Enter annotation: ')
        annotations.append(annotation)
        # Draw the point on the image
        cv2.circle(image, (x, y), 5, (0, 255, 0), -1)
        # Display the updated image
        cv2.imshow('Image', image)

# Set the callback function for mouse events
cv2.setMouseCallback('Image', annotate)

# Wait for the user to annotate the image
cv2.waitKey(0)

# Print the annotated points and annotations
for point, annotation in zip(points, annotations):
    print(f'Point: {point}, Annotation: {annotation}')

# Save the annotated image
cv2.imwrite('annotated_image.jpg', image)

# Close the image window
cv2.destroyAllWindows()
```

Figure 1: The output ChatGPT generated when prompted by "Python-based image annotation program."

```python
from PIL import Image, ImageDraw
import tkinter as tk

# Initialize the GUI
root = tk.Tk()

# Load the image
image_path = 'image.jpg'
image = Image.open(image_path)

# Create a canvas to display the image
canvas = tk.Canvas(root, width=image.width, height=image.height)
canvas.pack()

# Convert the PIL image to Tkinter-compatible format
image_tk = ImageTk.PhotoImage(image)
canvas.create_image(0, 0, anchor=tk.NW, image=image_tk)

# Create lists to store annotation coordinates
points = []
annotations = []

# Callback function for mouse events
def annotate(event):
    # Add the clicked point to the list
    points.append((event.x, event.y))
    # Prompt the user to input annotation
    annotation = input('Enter annotation: ')
    annotations.append(annotation)
    # Draw the point on the canvas
    canvas.create_oval(event.x - 5, event.y - 5, event.x + 5, event.y + 5, fill='green')
    # Refresh the canvas
    root.update()

# Bind the callback function to the canvas mouse click event
canvas.bind('<Button-1>', annotate)

# Start the GUI event loop
root.mainloop()

# Print the annotated points and annotations
for point, annotation in zip(points, annotations):
    print(f'Point: {point}, Annotation: {annotation}')

# Draw the annotated points on the image
draw = ImageDraw.Draw(image)
for point in points:
    draw.ellipse((point[0] - 5, point[1] - 5, point[0] + 5, point[1] + 5), fill='green')

# Save the annotated image
annotated_image_path = 'annotated_image.jpg'
image.save(annotated_image_path)

print('Image annotation complete.')
```

Figure 2: A revised program that contrasts from the earlier program that depended on OpenCV. The most notable differences occur near the beginning, when importing the PIL and Tkinter libraries.

guidance, such as an outline of the requirements or an initial excerpt. Ultimately, the programmer was again unable to use ChatGPT to produce working code.

Attempt 3:
The third attempt once again changed the order in which the features were requested in hopes of producing a different outcome, starting with the input *"Python-based image annotation code. Can you modify the above code to run without using OpenCV?"*, followed immediately by a request to include an annotation coordinate-tracking feature. The chat-bot's reply (Figure 4) included parts of code that intended to fulfill the desired features.

While the app would open, the user's image would not upload, and the annotations on the canvas were invisible to the user. After some repetitive requests to fix the errors, it was determined that ChatGPT could not produce a working code.

Method Takeaway: Overall, the Scratch Evolution method was not effective. ChatGPT struggled to output the original code that functioned and to manipulate the initial code while preserving the program's functionality.

2.2 Method 2: From Sample

This method begins by providing ChatGPT with the code for a previously developed GUI [4] that produces a Python app (Figure 5) that functions but does not have all the required features. For example, the program contains no zoom feature, no functioning save feature, and an eraser that undesirably erases the image in

```python
from PIL import Image, ImageDraw
import tkinter as tk

# Initialize the GUI
root = tk.Tk()

# Load the image
image_path = 'image.jpg'
image = Image.open(image_path)

# Create a canvas to display the image
canvas = tk.Canvas(root, width=image.width, height=image.height)
canvas.pack()

# Convert the PIL image to Tkinter-compatible format
image_tk = ImageTk.PhotoImage(image)
canvas.create_image(0, 0, anchor=tk.NW, image=image_tk)

# Create lists to store annotation coordinates
points = []
annotations = []

# Callback function for mouse events
def annotate(event):
    # Add the clicked point to the list
    points.append((event.x, event.y))
    # Prompt the user to input annotation
    annotation = input('Enter annotation: ')
    annotations.append(annotation)
    # Draw the point on the canvas
    canvas.create_oval(event.x - 5, event.y - 5, event.x + 5, event.y + 5, fill='green')
    # Refresh the canvas
    root.update()

# Bind the callback function to the canvas mouse click event
canvas.bind('<Button-1>', annotate)

# Start the GUI event loop
root.mainloop()

# Print the annotated points and annotations
for point, annotation in zip(points, annotations):
    print(f'Point: {point}, Annotation: {annotation}')

# Draw the annotated points on the image
draw = ImageDraw.Draw(image)
for point in points:
    draw.ellipse((point[0] - 5, point[1] - 5, point[0] + 5, point[1] + 5), fill='green')

# Save the annotated image
annotated_image_path = 'annotated_image.jpg'
image.save(annotated_image_path)

print('Image annotation complete.')
```

Figure 3: A revised version of the earlier program that includes the annotation recording feature.

```python
from PIL import Image, ImageDraw
import tkinter as tk

def draw_rectangle(event):
    global top_left_pt, bottom_right_pt, drawing
    x = event.x - canvas_width // 2
    y = canvas_height // 2 - event.y

    if event.type == '4':  # ButtonPress event
        drawing = True
        top_left_pt = (x, y)

    elif event.type == '5':  # ButtonRelease event
        drawing = False
        bottom_right_pt = (x, y)
        draw.rectangle([top_left_pt, bottom_right_pt], outline='green')
        image.show()

def display_coordinates(event=None):
    center_x = canvas_width // 2
    center_y = canvas_height // 2
    top_left_x = top_left_pt[0] + center_x
    top_left_y = center_y - top_left_pt[1]
    bottom_right_x = bottom_right_pt[0] + center_x
    bottom_right_y = center_y - bottom_right_pt[1]
    print(f"Top Left: ({top_left_x}, {top_left_y})")
    print(f"Bottom Right: ({bottom_right_x}, {bottom_right_y})")

image_path = "path/to/image.jpg"
image = Image.open(image_path)
canvas_width, canvas_height = image.size
top_left_pt, bottom_right_pt = (-1, -1), (-1, -1)
drawing = False

root = tk.Tk()
root.title("Image Annotation")
canvas = tk.Canvas(root, width=canvas_width, height=canvas_height)
canvas.pack()

draw = ImageDraw.Draw(image)
photo_image = ImageTk.PhotoImage(image)
canvas.create_image(canvas_width // 2, canvas_height // 2, image=photo_image)

canvas.bind("<ButtonPress-1>", draw_rectangle)
canvas.bind("<ButtonRelease-1>", draw_rectangle)
root.bind("c", display_coordinates)

root.mainloop()
```

Figure 4: ChatGPT's original code, prompted by the user to create a Python-based image annotation code that did not use OpenCV and included an annotation coordinate-tracking feature.

addition to the markups. Depending on the window size, not all of the toolbar may be accessible.

Using this code as a baseline, ChatGPT further developed the code to satisfy the requirements of the segmentation program.

Attempt 1:

The initial input prompt was *"What is the function of this program?"*. ChatGPT replied with a walk through each part of the provided code. Additionally, a concluding overview of the program was as follows:

ChatGPT:
Overall, the program provides a basic image markup tool where users can draw on an image, choose different tools and colors, adjust brush size, clear the canvas, and save the modified image.

From here, the programmer was then able to gradually request the implementation of new features to fit the requirements of our desired program. While ChatGPT can recognize the sample code initially given to it and can make modifications to implement the requested features, it does a poor job of keeping the code functional. Therefore, it is essential to recurrently ask the chat-bot to check for bugs after each new addition to the program to prevent an accumulation of issues over time. One consistent issue to note when working with ChatGPT is that even the requested corrections may not work. For instance, consider the zoom-in and zoom-out features. The initial requests are as follows:

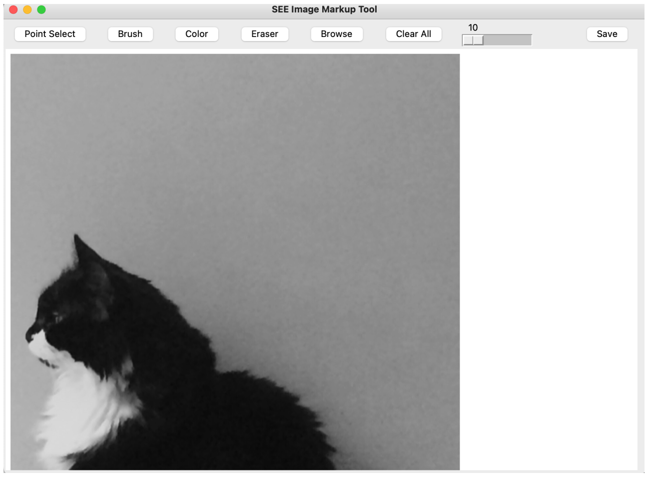

Figure 5: Screenshot of the Python app generated by the sample code.

User:
Is there a way you can manipulate the earlier code to integrate a zoom functionality into the program? For instance, can you add a "+" button for the user's view of the canvas to zoom in and a "-" button to zoom out? The purpose is to make it easier for the user to see the details of the image only when the view is zoomed in and the overall image only when the view is zoomed out?

ChatGPT:
Certainly! Here's an updated version of the code that integrates zoom functionality with "+" and "-" buttons for zooming in and out of the canvas: . . .

Although the chat-bot's reply seems promising, after some testing, it was noticed that the "+" and "-" buttons are present but do not zoom in and out. ChatGPT was prompted to address this issue:

User:
The program zoom buttons do not seem to work. Is there any way you can make some modifications to ensure the logic created enables them to serve their intended purpose?

ChatGPT:
Apologies for the oversight. It seems we missed updating the zoom functionality. Here's the modified code with corrected zoom functionality: . . .

The provided modifications did not address the issue. The conversation continued for several iterations of user input and chat-bot output, resulting in minor tweaks to the program until the buttons functioned. The chat-bot frequently recommended creating custom buttons that had many initial errors. Yet, once the buttons were functional, the features previously established no longer worked properly. The final product (Figure 6) contained buttons that were accessible but did not function. While the zoom buttons work, their activation resulted in the erasing of annotations. Additionally, the annotations do not align with the mouse and may occasionally become dysfunctional. Only PNG images can be uploaded, and there is no save feature.

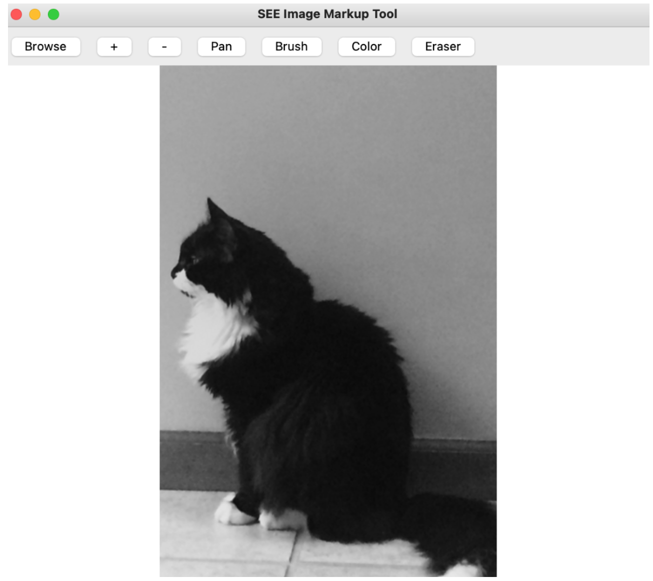

Figure 6: Screenshot of the Python app generated by the updated code. Note that the program now has zoom and pan features.

Method Takeaway: Overall, the Sample method was not effective. Due to the conversational nature of working with ChatGPT, it is exceedingly difficult to pinpoint one cause for the ineffectiveness. ChatGPT's learned methods may fail to integrate with the sample code given. The system may find it challenging to process large amounts of code upfront.

2.3 Method 3: From Scratch List Requirements

The method titled "Scratch List Requirements" starts with entering an extensive list of requests and goals into ChatGPT to create an image annotation program in Python. The revised list responds to the issues persisting in the "From Sample" method. Similarly to the previous method, this method used the list of requirements to create an initial code that acts as a baseline. After spotting the errors in the initial program, the programmer focused on fixing individual features one at a time with assistance from ChatGPT to create a functioning program that satisfies all of the listed requirements. After an unsuccessful first attempt, the second attempt depends on a slightly different initial list.

Attempt 1:
The initial user input included a list of program requirements written for the first attempt.

User:
Create a Python-based image annotation code satisfying the following conditions: Do not use `CV2`*, Instead of using* `ANTIALIAS`*, use* `LANCZOS`

or `Resampling.LANCZOS`*, Include a prompt that enables the user to select an image from their local files, Save both the original image and the annotated copy of the image and include a place for the user to choose the location at which the annotated copy will be saved, Enable the user to annotate the image by interacting using a computer mouse, The annotation tools include a black marker, white marker and annotation eraser that have the ability to draw atop the pixels in an image. The user can change the size of the tools throughout their session, with the smallest size being one pixel and the largest size being 20 pixels. Create a coordinate system for the image that look at the pixel location of the annotations, and in which the center of the image is the origin, regardless of its placement on the user's screen, Create a document that records a list of these coordinate points. Prompt the user to choose a file location at which to save this document. Double-check that the code has no errors, by making sure the GUI event loop has* `root.mainloop()` *and that all of the proper packages are imported during initialization.*

In response, ChatGPT formulated a program that was not functional and produced error messages, as follows. When asked what adjustments would be required to get the code running, ChatGPT supplied a bullet point list and, upon request, offered up code and implemented it into the pre-existing program. Although the chatbot can spot errors in its code, it continues to output error-ridden code. Eventually, the programmer was able to get the program (Figure 7) up and running. In this version, most of the requested features are present with flaws, except for the upload image function, which is essential to the program. The program can only annotate in dots and has no scroll functionality. Depending on the window size, some of the features may not be accessible.

Figure 7: Screenshot of the initial GUI generated by Method 3, in which a list of requirements was provided.

ChatGPT generates the initial code very quickly. The authors were curious if there would be a way to ask the chat-bot to take more time in its code generation to check for errors throughout. However, GPT-4 takes more time to type the response, and as will be discussed in Method 5, the resulting outputs have higher quality. Although the system can add comments to its code to help the user follow along, one of the limitations of ChatGPT is that there is no way to see the logic behind the output or the sources behind the information.

When focusing on one specific element of the code, ChatGPT frequently makes alterations that result in the dysfunction of the others. After much discussion with the chat-bot, the code (Figure 8) contained fewer functioning features than the initially generated code, albeit with no error messages.

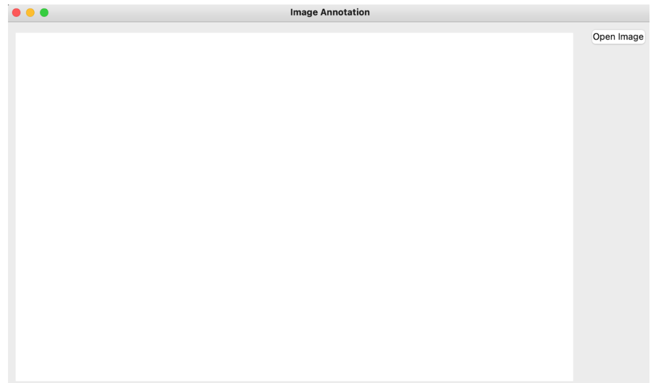

Figure 8: Screenshot of the final GUI generated by Method 3 - Attempt 1. Note that there is no 'save image' feature.

Attempt 2:
For the second attempt, the programmer created a list of more detailed requirements. However, this large amount of input information revealed a limitation of ChatGPT, wherein the chatbot conveyed its limit when taking in long and complicated requests:

User:
Create a GUI for image annotation that satisfies the following conditions: . . .

ChatGPT:
Creating a complete GUI application with all the described features is beyond the scope of a text-based response. However, I can provide you with an outline of the steps involved in building such an application using Python and some popular GUI libraries. To create the GUI for image annotation with the specified features, you can follow these steps: . . .

The term "text-based response" is most interesting as it was able to form large programs as output during prior attempts and methods.

With some prompting, ChatGPT generated a sample of code for the image annotation program. When asked to review its code, the system identified unfilled parts of the listed requirements. Each ChatGPT-generated code was input as new code to be corrected. At one point, ChatGPT even asked for more specific instructions to best direct its alterations. Eventually, to increase specificity, the chat-bot was requested to create a program (Figure 9) with three tabs; the first one had annotating features such as panning and zooming, the second one was a continuously updated view of the annotated image, and the third tab was a blank window that appeared to serve no purpose.

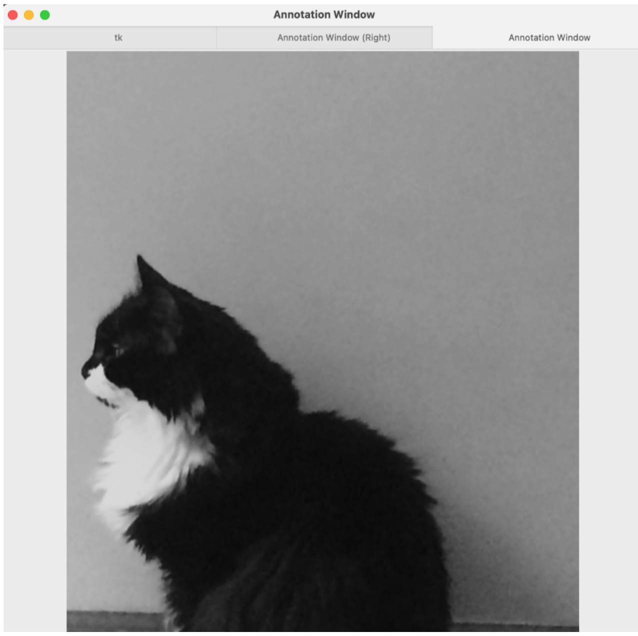

Figure 9: Screenshot of the GUI generated by Method 3 - Attempt 2. Note that multiple tabs are generated.

With the final layout, the toolbar was not accessible so that the tabs could not serve their designated purposes.

Method Takeaway: Overall, the "From Scratch List Requirements" method was ineffective as it provided too much information upfront for the system to process. Despite the clarity of the input given, ChatGPT failed to provide output that took all elements of the request into account.

2.4 Method 4: From Scratch Assisted

The method titled "From Scratch Assisted" starts with entering a rough baseline code as created through user knowledge gained throughout this study and prior methods. This code generated from scratch by the user does not meet all of the specifications required or run and will be corrected and further developed through recommendations by ChatGPT.

Attempt 1:
The first attempt used Tkinter [13] in a program that contained unresponsive `'Browse'`, and `'Clear All'` buttons; black and white brushes; tool size slider; and zoom and pan features. As such, the GUI was nonfunctional. The general process was to suggest a new feature, debug, and then evaluate the new code based on prior knowledge. Once the code demonstrated some functionality or comprehensible logic by having a few error codes, a new feature would be suggested. ChatGPT successfully interpreted error codes, such as the one included in Figure 10 while also providing explanations when requested.

When given this text-dense error message, ChatGPT responds as follows:

ChatGPT:
The error message `TclError: cannot use geometry manager pack inside . which already has slaves managed by grid` *indicates that you are trying to mix the pack and grid geometry managers in your code, which is not allowed.*
In your code, you are using both grid and pack to manage widgets within the same parent widget. In the init method, you have used a grid to place widgets in the root window. However, later on, you are using pack to manage the annotation_canvas widget.
To fix this error, you need to be consistent and use either grid or pack for all widgets within a given parent widget. In this case, since you are already using grid to manage other widgets, you should use grid for the annotation_canvas widget as well.

The chat-bot also provided an updated code meant to address the issue. One current limitation of ChatGPT is the inability to highlight within blocks of provided code. When requesting that ChatGPT modify pre-written code, it can be challenging to pick out the changes the chat-bot made. Upon request, comments inside the code indicate the changes, but even those can be difficult to spot when considering lengthy blocks of code.

Another feature of ChatGPT is its ability to recall elements from earlier parts of the conversation. For instance, after conversing back and forth about specific flaws in an earlier block of code to suggest changes, the chat-bot implemented the suggestions into the original code. The response was characterized by a comparable level of confidence and positivity:

ChatGPT:
Certainly! Here's the complete code for the image annotation tool based on the discussions we've had…

The final product (Figure 11) had all the required buttons except for a save button; however, the buttons did not successfully activate any features or annotation tools upon selection, leading the GUI to be ultimately ineffective.

Attempt 2:
The second attempt used Plotly Dash [11] in a program that, similar to the one used in Attempt 1, lacked the capability to upload images and had non-functional buttons. The initial code was constructed from scratch drawing from pieces of code gathered throughout previous methods and inspired by examples of programs with similar features.

After using the chat-bot to fix the code and get it running (Figure 12), the layout included a full toolbar that remained accessible regardless of the window size. The markups became erased whenever the tool color or size was changed. The saved image had a white, gridded background and did not retain the same size as the original. In addition, it still lacked the buttons to trigger the requested features. ChatGPT suggested JavaScript to create custom buttons, but they remained dysfunctional. However, the chat-bot seemed reluctant to offer Python alternatives to the JavaScript buttons.

Method Takeaway: Overall, the programmer found this method to be most effective. ChatGPT struggles to produce code from scratch and sometimes uses logic that can be challenging to follow.

```
---------------------------------------------------------------------------
TclError                                  Traceback (most recent call last)
Cell In[1], line 197
    193             self.zoom_factor = min(width_ratio, height_ratio)
    195             self.update_canvas()
--> 197 Image_Markup()

Cell In[1], line 56, in Image_Markup.__init__(self)
     53 self.setup()
     55 self.annotation_canvas = tk.Canvas(self.root)
---> 56 self.annotation_canvas.pack(side = tk.LEFT, fill = tk.BOTH, expand = True)
     58 self.annotation_canvas.bind("<ButtonPress-1>", self.start_pan)
     59 self.annotation_canvas.bind("<B1-Motion>", self.pan_image)

File ~/anaconda3/lib/python3.10/tkinter/__init__.py:2425, in Pack.pack_configure(self, cnf, **kw)
   2409 def pack_configure(self, cnf={}, **kw):
   2410     """Pack a widget in the parent widget. Use as options:
   2411     after=widget - pack it after you have packed widget
   2412     anchor=NSEW (or subset) - position widget according to
   (...)
   2423     side=TOP or BOTTOM or LEFT or RIGHT -  where to add this widget.
   2424     """
-> 2425     self.tk.call(
   2426           ('pack', 'configure', self._w)
   2427           + self._options(cnf, kw))

TclError: cannot use geometry manager pack inside . which already has slaves managed by grid
```

Figure 10: A sample of the error code given to, and successfully deciphered by, ChatGPT.

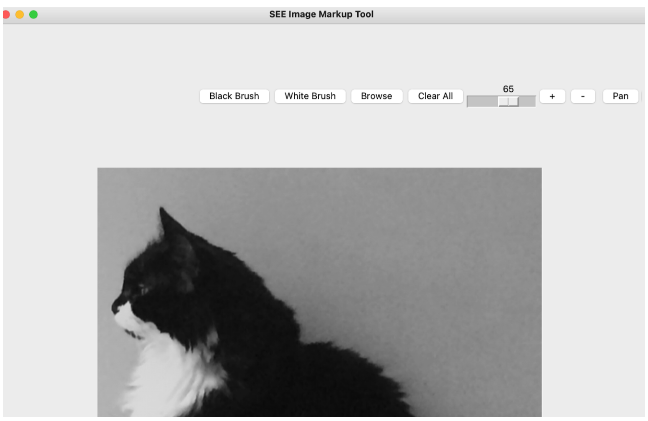

Figure 11: Screenshot of the GUI generated by Method 4 - Attempt 1. Note that the toolbox may not be accessible if the window size is too small.

As such, it is easier for the programmer to start with a clear road map in mind and a rough draft of code.

2.5 Method 5: Using GPT-4

In this methodology, the programmer initially requested ChatGPT to develop a GUI based on a specified set of requirements, though not exhaustively. Subsequently, the generated code was tested, followed by iterative refinements and enhancements, including the addition of new features, aligning this approach with Method 1.

Initially, the GUI encountered an issue where the displayed image was disproportionately small. The GUI, embedded within a Python window, was then transformed into a web-based interface, utilizing Python with Flask [12], HTML, and JavaScript. A critical feature involved the transmission of x- and y-coordinates from user-drawn points to Python. Additionally, the application was designed to overlay an annotation mask on the input image, enabling users to

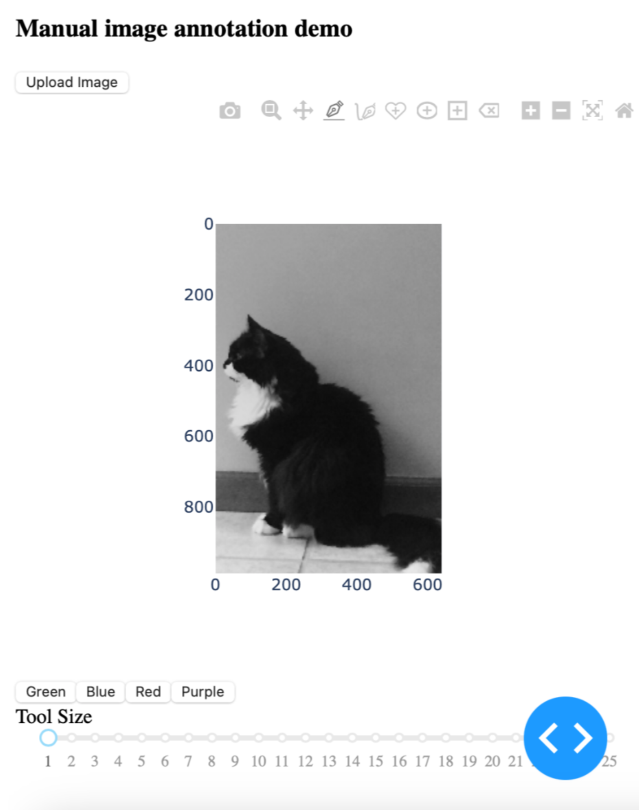

Figure 12: Screenshot of the GUI generated by Method 4 - Attempt 2.

create segmentation annotations without altering the original image. This functionality involved dynamic mask updates, including real-time display refreshment upon user interaction, and the capacity to save the mask as a separate `PNG` file. Therefore, the essential features incorporated were: (a) Utilization of two image elements – the original base image and the overlay (mask); (b) Incorporation of a canvas for drawing on the overlay; (c) Functionality to save the overlay as a new image; (d) Dynamic updating of the mask in response to user edits.

Despite the successful integration of these features, the GUI initially lacked a `'save'` button. After multiple iterations, it was discerned that the issue stemmed from the stacking order of HTML elements, with the canvas inadvertently obscuring the button. This realization was attributed to the programmer's prior experience, underscoring the significance of foundational programming knowledge in resolving such issues.

The final refinement involved addressing an issue where the GUI would connect the end point of one drawing segment to the start point of the next upon resumption. ChatGPT resolved this by modifying the `mouseup` event handler to invoke the `beginPath()` method. This adjustment ensured that new drawing segments initiated after releasing the mouse button did not connect with the preceding segments, thereby enhancing the application's functionality and user experience.

Figure 13 presents a screenshot of the rudimentary GUI developed with Method 5. The interface of the GUI is minimalistic, primarily displaying the image in its original dimensions alongside a `'Save Mask'` button situated beneath the image. Users have the capability to annotate directly on the image; these annotations, appearing in red, are made on a supplementary canvas layer above the image, overlaying it without altering the original. Upon clicking the `'Save Mask'` button, only the annotated mask is preserved.

This version of the GUI is intentionally basic, devoid of advanced functionalities such as zooming capabilities, varied pen colors for annotating distinct objects, or the ability to fill enclosed areas in the annotations. Its primary purpose is to facilitate a comparative analysis with the GPT-3.5 generated interfaces and to demonstrate the relative ease and efficiency of GUI generation using GPT-4, which required fewer iterations and refinements in requests.

Potential enhancements could include instructing the chat-bot to fill the annotated areas with a specific color, such as white, while rendering the background in black, thereby further refining the GUI's functionality.

Figure 13: Screenshot of the GUI generated by Method 5. This figure includes the original image (left), the user-annotated image (middle), their respective save buttons below each, and the saved mask after pressing the save button (right).

Method Takeaway: This method demonstrates a level of comparability to Method 1, wherein GPT-3.5 was utilized. The application of GPT-4 in this context revealed a marked improvement in the chat-bot's comprehension capabilities, leading to the fulfillment of requirements with fewer follow-up interactions and iterations. The code produced by GPT-4 was notably effective and met the desired criteria satisfactorily.

However, the experience and prior knowledge of the programmer played a pivotal role in identifying the mentioned issue that ChatGPT was seemingly unable to detect. Consequently, this underscores that sole reliance on ChatGPT, in the absence of foundational programming knowledge, is not advisable.

3 DISCUSSION AND RECOMMENDATIONS

Generally, leveraging prompt engineering with large language models like ChatGPT possibly leads to significantly improved outcomes. By thoughtfully designing prompts, users can more precisely steer the model towards producing the desired responses.

To reduce misunderstandings between the programmer and the chat-bot, specific instructions to ChatGPT are advisable. The programmer should maintain some skepticism regarding the output. While the chat-bot delivers responses that exude confidence, its output remains occasionally incorrect. It is recommended to have a well-defined objective: similar to the broader practice of coding, engaging with ChatGPT yields optimal results when the user approaches with a specific intention in mind. ChatGPT seems to provide the best responses when user requests are specific.

Either start small or break the code down.
ChatGPT typically exhibits limitations when processing large requests, often resulting in the omission of certain parts. It is therefore recommended to initiate interaction with either a small request or an initial draft, subsequently refining it piece by piece. The most effective strategy is contingent upon the nature of the project. If the program has interdependent components, starting with basic features and progressively elaborating them is advisable. If the program is characterized by sequential components, it would be more effective to break down the problem and refine it step-by-step.

Pay attention to the modifications made by ChatGPT and ask questions to understand the logic.
While ChatGPT is capable of analyzing large text inputs, it may not thoroughly address every aspect of the code. Providing suggestions is more beneficial than code corrections, as they can provide starting points for further investigation. Additionally, a compiled list of recommendations may be more comprehensible than suggestions embedded within the code. As stated earlier, ChatGPT cannot be highlighted within a Python code box. While it can insert comments, such annotations can be challenging to distinguish from comments input by the user.

4 CONCLUSION

ChatGPT can be a supportive tool for novice programmers. While the chat-bot is not adept at generating highly functional code from scratch, it excels in identifying errors and interpreting the logic of existing programs. Due to its conversational nature, ChatGPT is particularly effective at responding to specific questions. In general, the practice of refining prompts through prompt engineering with this advanced language model probably enhances the result quality. By carefully crafting and adjusting prompts, users can direct the model more effectively toward generating outputs that meet their specific requirements.

This study merely scratched the surface of ChatGPT's capacity to aid in user interface programming. A more granular examination of specific methods might enable programmers to discern patterns in the system's responses and check for replaceable conversations.

Additionally, a programmer with experience in graphical user interface development is better positioned to evaluate the potential of chat-bots in enhancing their productivity in GUI design.

REFERENCES

[1] 2024. ChatGPT — Release Notes | OpenAI Help Center. https://help.openai.com/en/articles/6825453-chatgpt-release-notes. Accessed on 3/8/2024.

[2] Katharina Buchholz. 2023. Infographic: Threads Shoots Past One Million User Mark at Lightning Speed. https://www.statista.com/chart/29174/time-to-one-million-users/. Accessed: 2023-11-12.

[3] Enrico Checcucci, Paolo Verri, Daniele Amparore, Giovanni E. Cacciamani, Cristian Fiori, Alberto Breda, and Francesco Porpiglia. 2023. Generative Pre-training Transformer Chat (ChatGPT) in the scientific community: the train has left the station. *Minerva Urology and Nephrology* 75, 2 (April 2023), 131–133. https://doi.org/10.23736/S2724-6051.23.05326-0

[4] Dirk Colbry. 2022. GUI-Basic. https://github.com/see-insight/see-tkinter-gui/blob/dd5c50c5bb0ca92f63cc9b48b85230c2a5d65311/GUI-Basic.py.

[5] Dirk Colbry. 2023. Toward an automatic exploration of algorithm space to speed up image annotation for applications in scientific image understanding. In *IEEE AIPR workshop*. St. Louis, MO.

[6] Jeffrey A. Clark (Alex) and contributors. 2024. Pillow - Python Imaging Library (Fork). https://pypi.org/project/Pillow/. Accessed on 3/8/2024.

[7] Gary Marcus. 2022. What to Expect When You're Expecting … GPT-4. https://garymarcus.substack.com/p/what-to-expect-when-youre-expecting. Accessed: 2023-11-12.

[8] Dan R. Olsen. 2007. Evaluating user interface systems research. https://dl.acm.org/doi/10.1145/1294211.1294256. In *Proceedings of the 20th annual ACM symposium on User interface software and technology (UIST '07)*. Association for Computing Machinery, New York, NY, USA, 251–258. https://doi.org/10.1145/1294211.1294256

[9] OpenAI. 2024. OpenAI. https://openai.com. Accessed on 3/8/2024.

[10] OpenCV. 2024. OpenCV - Open Computer Vision Library. https://opencv.org. Accessed on 3/8/2024.

[11] Plotly Dash. 2024. Dash Python User Guide. https://dash.plotly.com. Accessed on 3/8/2024.

[12] Armin Ronacher. 2024. Flask Web Framework. https://flask.palletsprojects.com/. Accessed on 3/8/2024.

[13] Tkinter. 2024. tkinter — Python interface to Tcl/Tk. https://docs.python.org/3/library/tkinter.html. Accessed on 3/8/2024.

APPENDIX

Here we present the final, fully functional version of the code generated in Method 5 using GPT-4, to facilitate replication and further research by interested parties. This includes the `app.py` file, a Flask [12] application, and the accompanying `index.html`.

In `app.py`, we employ the `render_template('index.html', form=form)` function, which integrates Flask with the HTML frontend, showcasing the interaction between backend logic and frontend presentation. This setup exemplifies the practical application of AI-generated code in a web development context.

app.py:

```python
from flask import Flask, render_template, request, jsonify, redirect, url_for
from flask_wtf import FlaskForm
from wtforms import SubmitField
from PIL import Image
import os
import base64
from io import BytesIO

app = Flask(__name__)
app.config['SECRET_KEY'] = 'mysecretkey'

class SaveMaskForm(FlaskForm):
    save = SubmitField('Save Mask')

@app.route('/', methods=['GET', 'POST'])
def index():
    form = SaveMaskForm()
    if form.validate_on_submit():
        # The functionality of saving the mask will be handled in the JS.
        # This redirect is just to reload the form after the POST request.
        return redirect(url_for('index'))
    return render_template('index.html', form=form)

@app.route('/save_mask', methods=['POST'])
def save_mask():
    data = request.json
    mask_data = data['mask_data']

    mask_img = Image.open(BytesIO(base64.b64decode(mask_data.split(',')[1])))
    mask_img_name = "flower_mask.png"
    mask_img.save(os.path.join('static', mask_img_name))

    return jsonify(status='success')

if __name__ == "__main__":
    app.run(debug=True)
```

index.html:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Image Annotation</title>
    <style>
        #control-panel {
            margin-top: 20px;
        }
    </style>
</head>
<body>

<img id="sourceImage" src="/static/flower.png" style="position: relative; z-index: 0;" alt="Source Image">
<canvas id="myCanvas" style="position: absolute; top: 0; left: 0; z-index: 1;"></canvas>

<div id="control-panel">
    <button type="button" onclick="saveMask()">Save Mask</button>
</div>

<script>
    let canvas = document.getElementById('myCanvas');
    let ctx = canvas.getContext('2d');
```

```
    let img = document.getElementById('sourceImage');

    let isDrawing = false;

    img.onload = function() {
        canvas.width = img.width;
        canvas.height = img.height;
    }

    canvas.addEventListener('mousedown', function(event) {
        isDrawing = true;
        draw(event.pageX - canvas.offsetLeft, event.pageY - canvas.offsetTop);
    });

    canvas.addEventListener('mousemove', function(event) {
        if (isDrawing) {
            draw(event.pageX - canvas.offsetLeft, event.pageY - canvas.offsetTop);
        }
    });

    canvas.addEventListener('mouseup', function() {
        isDrawing = false;
        ctx.beginPath();  // This line clears the current drawing path
    });

    function draw(x, y) {
        ctx.lineWidth = 5;
        ctx.lineCap = 'round';
        ctx.strokeStyle = 'red';

        ctx.lineTo(x, y);
        ctx.stroke();
        ctx.beginPath();
        ctx.moveTo(x, y);
    }

    function saveMask() {
        const canvasData = canvas.toDataURL();
        fetch('/save_mask', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                'mask_data': canvasData
            })
        })
        .then(response => response.json())
        .then(data => {
            if (data.status === 'success') {
                alert("Mask saved successfully!");
            } else {
                alert("Error saving mask!");
            }
        });
    }
</script>

</body>
</html>
```