Name: Sravani Pati **Prof: Paige Anne Rodeghero**

CUID: C79908900 **CPSC 8740**

AI-Receptive Software Development

1. Setup and Introduction

Artificial Intelligence is now a major part of our world. It is used in tools we interact with every day, from search engines to mobile apps, and especially in **software development**. But while AI is powerful, it is also important to remember that it does nothing on its own. It is not a replacement for human thinking. It does not build or create anything unless we give it purpose. The true strength of AI comes when we use it wisely with our own ideas, logic, and creativity.

For this project, I selected **ChatGPT** as my main AI tool. I used Visual Studio Code as my development environment to write, run, and test all the projects. There are many other tools like **GitHub Copilot** and **other code assistants**, but I chose **ChatGPT** because it fits perfectly with how I work. It is not just a code generator. It is a thinking partner. I can share my ideas, ask technical questions, brainstorm solutions, and get help when I get stuck. I use ChatGPT in my daily life as a researcher as well. Anytime I have an idea, I drop it into ChatGPT. I ask what can be improved, how something can be done better, and sometimes I just ask questions to learn more about a topic I am exploring.

It has helped me understand things more deeply. But one thing I always remember is that I do not accept the first answer blindly. What I need or expect will not always come in the first response. It is a conversation. I ask follow-up questions, I test the code, I check what is working and what is not. Through multiple iterations, I begin to understand how ChatGPT is approaching a problem. That process of back-and-forth helps me grow not only as a developer but also as a learner. ChatGPT does not give the final solution. It gives possibilities. It is up to me to make the right decisions.

In this project, I used **Python** and the **Tkinter** library to create three applications. Each project was unique and needed different types of logic and design.

The first was a **smart calculator**. I wanted this to go beyond just basic math. So I built a system that could visualize equations, show number lines, provide math hints, and even suggest the next step in solving a problem. ChatGPT helped me build the layout, improve the logic using libraries like **sympy** and **matplotlib**, and refine how the results were shown.

The second was a **to-do list** application. But instead of something simple, I created a full productivity app. It had task filtering, overdue task tracking, a focus timer, light and dark modes,

and even an analytics dashboard with charts. ChatGPT helped with organizing the code, designing the UI, and even suggested features I had not thought of at first.

The third project was a **tic-tac-toe** game. This was built with a series mode, AI opponents at different difficulty levels, visual effects, and hints for players. ChatGPT helped me understand how to implement the minimax algorithm for AI and how to manage the game state with series tracking. It was a great learning experience, especially with debugging the AI logic and making sure the user experience was smooth.

From all these tasks, I can say with confidence that AI tools like ChatGPT do not remove the need for understanding. In fact, they require it. If I do not understand what I am asking or what it is responding with, then the tool is not being used properly. But if I take time to explore its answers, ask the right questions, and build my own knowledge through the process, then it becomes incredibly powerful.

This project was a clear example of that. I combined my own **problem-solving** with the assistance of **ChatGPT** to build complete working applications. I did not just copy responses. I learned from them. And that is what makes AI a real asset not as something that replaces us, but something that enhances the way we think, work, and create.

2. Experimentation

Working on this project was not just about finishing tasks. It was about learning how to work with AI in a real and meaningful way. I brought my own ideas to each app and used ChatGPT in Visual Studio Code to explore options, ask questions, build drafts, and improve them. I did not take the first answer at face value. I tested, compared, and iterated until the results matched my goals. I treated ChatGPT like a collaborator, studied its suggestions, fixed mistakes, and refined the design through multiple rounds. The sections that follow show how this process shaped the Smart Visual Calculator, the To Do app, and Tic Tac toe game.

Smart Visual Calculator

I created the Smart Visual Calculator with one goal in mind, to solve problems while teaching the logic behind them. At first glance it looks simple, with an input box, keypad, and toolbar, but behind that clean interface there is a system designed to both calculate and explain. Each expression is broken down into tokens, processed safely with SymPy, and then brought to life through visual models. I even designed it to understand natural phrases like divided by or multiplied by so it can adapt to the way people naturally think about math.I built this entire application in Python using the Tkinter library inside Visual Studio Code.

Name: Sravani Pati **Prof: Paige Anne Rodeghero**

CUID: C79908900 **CPSC 8740**

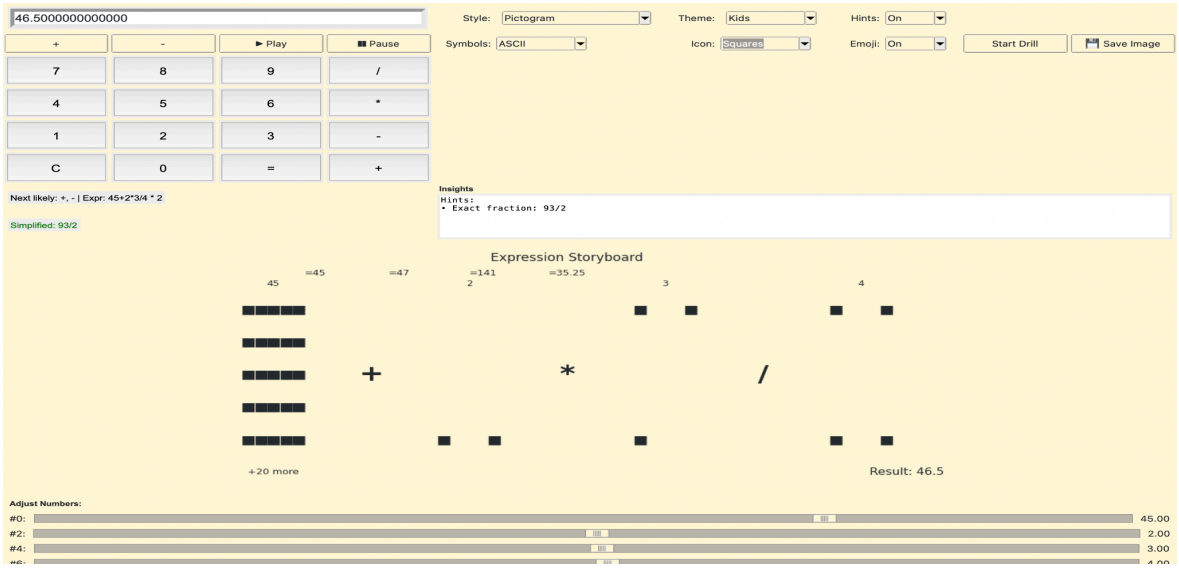


Figure 1: Smart Visual Calculator

The most distinctive feature is the visual storyboard. Instead of showing only a number, the calculator displays the full process step by step. Numbers appear as pictograms with dots, stars, hearts, or emoji, operators are positioned clearly between them, and intermediate results are displayed along the way before the final answer appears. To make concepts even clearer I added specialized models. Multiplication can be seen as a grid of rows and columns, division is represented as items placed into equal buckets, and addition and subtraction unfold as movement across a number line. These visuals make abstract operations far easier to grasp.

The calculator also feels intelligent and supportive. It remembers past operations and predicts the next likely step, showing suggestions as ghost buttons that users can click instantly. An insight engine analyzes each expression and provides hints such as simplified or expanded forms, exact fractions, and common factors. This creates a sense that the calculator is not only solving but guiding. I wanted users to interact directly with the math, so I added the ability to click on numbers in the canvas to edit them and watch the visualization update. Sliders below the canvas allow continuous adjustment of values, turning equations into an exploratory what if exercise. Animations add another layer, playing through each token one step at a time, while undo and redo keep every action safe.

Design and polish were also essential. The app includes three themes, Light, Dark, and Kids, and these styles extend into both the interface and the Matplotlib visuals so everything feels consistent. Icons can be swapped, emoji can be enabled, and any view can be saved as a PNG for later use. For practice I added a drill mode that detects which operator a user struggles with

most and generates problems to focus on that weakness, still offering predictions and hints during the session. Everything is held together with strong state management so calculations, visuals, hints, and animations always match. Errors like divide by zero are handled gracefully with a clear message. The final result is more than a calculator. It is a teaching companion that makes math visible, interactive, and engaging, supporting beginners who need intuition as well as experienced users who want a polished and capable tool.

While I would have liked to include detailed screenshots of each feature, I chose to keep this document focused and simple. Therefore, I have attached a full screenshot (Fig. 1) that captures the overall look and feel of the calculator, showcasing its visual design and key features.

Here is the link to the GitHub repository where the complete code for this calculator is available: Link : Github_link_for_smartcalculator_code

You can explore and interact with it exactly as described in this document. Just copy the files into your system, open them in Visual Studio Code, and run python3 smart_calculator.py in the terminal. You'll see the calculator in action and experience all its features step by step.

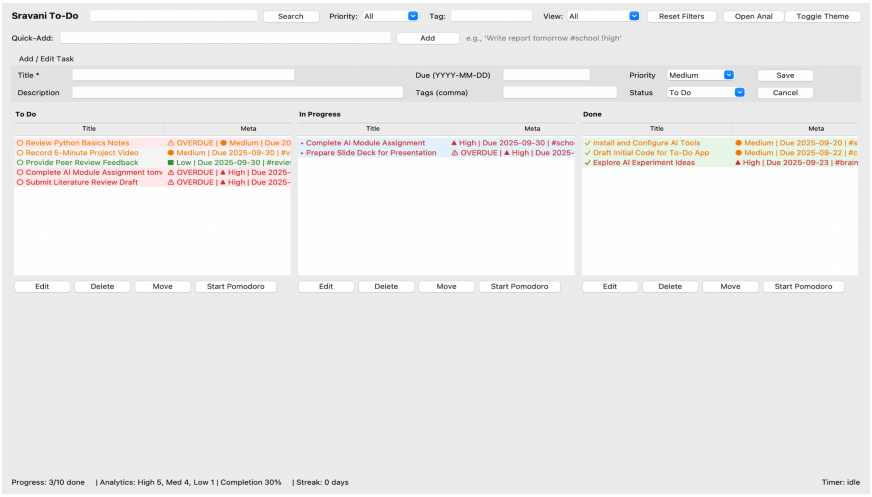
One of the challenges here was getting ChatGPT to handle symbolic math using SymPy, especially when I tried to mix natural language with equations. The early versions were often buggy or didn't run at all, but after several tries and re-explanations, we got it working smoothly.

To-Do List Application

I created the To-Do List application to be more than a basic task manager. It is built in Python with Tkinter inside Visual Studio Code, and it helps users plan, track, and improve their work with clarity. You can create, edit, and organize tasks with priority, due dates, tags, and status, and the board uses light and dark themes, color-coded rows, and clear status icons for quick scanning. Overdue tasks are highlighted with a red warning and a ⚠ icon so nothing slips by. A quick-add bar understands natural phrases like "Finish report tomorrow #work !high," and keyboard shortcuts keep everything fast. There is also a Pomodoro focus timer so you can lock in on one task without distractions.

Analytics are available in two places. The first is a live summary in the footer that updates as you work, showing totals, completion rate, and your daily streak of getting at least one task done. The second is the Open Analytics window, which presents three visual views that make progress feel tangible. One chart stacks tasks by status and priority so you can see where effort is concentrated. Another chart shows the overall priority mix at a glance. A third view groups

tasks by due timeline, such as overdue, today, the next seven days, later, or no date set. These visuals are simple, readable, and make the app feel genuinely insightful. Below screenshots shows the todolist application and analytics section. When I first asked ChatGPT for a to-do list, the code was simple and functional. I wanted something unique and more ambitious, so I started describing features like theme toggles, color-coded boards, quick-add parsing, a Pomodoro mode, and the two analytics layers. ChatGPT gave me drafts that sometimes had errors, and through several iterations of testing, fixing, and refining, the app grew into what I envisioned.



Here is the link to the GitHub repository where the complete code for this to-do list application is available: Link : Github_link_for_todolist. You can try it exactly as described here. Copy the files to your system, open them in Visual Studio Code, and run python3 todolist.py in the terminal to launch the full application.

Name: Sravani Pati **Prof: Paige Anne Rodeghero**

CUID: C79908900 **CPSC 8740**

Tic Tak Toe Application

I built Tic-Tac-Glow to make tic-tac-toe feel lively, fair, and replayable. It runs in Python with Tkinter inside Visual Studio Code and opens with a clean board, smooth glow effects, and an easy header where you can pick Two Players or Play vs Computer, choose who starts, set the difficulty to Easy, Medium, or Hard, and pick a series length such as best of 1, 3, or 7. The game now includes a Play Again button that appears at the end of each round so you can roll straight into the next game in the series. When a round ends you get a clear popup announcing the result, and when someone clinches the series you get a celebratory series-winner popup too.

The experience is designed to teach and guide. Hints can be turned on to show a subtle heatmap of good, draw, or losing squares, and a small coach panel lists the best predicted sequence of moves so players can see the logic unfold. There is also a Sandbox mode for trying "what if" moves without affecting the real game, plus an Undo, a Restart, and a Replay that animates every move from the last round. The AI plays automatically on its turn and scales from playful to perfect depending on the difficulty. You can personalize the feel with Light or Dark themes, switch piece styles between emoji, classic letters, or shapes, and even set custom emojis like a tiger for the player and a robot for the computer. A compact series panel on the right shows best-of settings, the running score, a row of per-game pips, and tiny bars for You, Computer, and Draw so the whole match is easy to follow at a glance. The screenshot for Fig. 3 highlights the board glow, the hints panel, the series tracker, and the top banner with mode, starter, difficulty, and series length, all in one view.

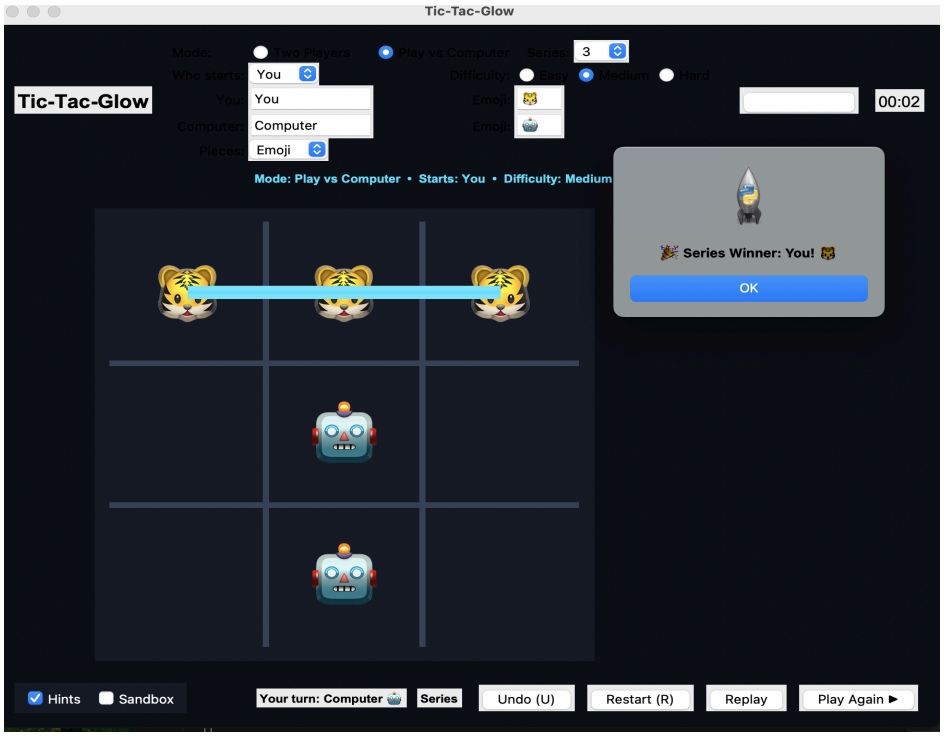
The quick-add input was more complex than I expected. Teaching ChatGPT to parse something like "Submit paper next Monday #school !high" into an actual task with date, tags, and priority took a lot of back-and-forth. But once we nailed the parsing logic, it really made the app feel smart.

Here is the link to the GitHub repository where the complete code for this game is available: Link: Github_link_for_tictactoe. You can try it exactly as described here. Copy the files to your system, open them in Visual Studio Code, and run python3 tictaktoe.py in the terminal to play, switch themes, toggle hints, experiment with Sandbox, and run a full best-of series with popups and celebrations.

The hardest part here was getting the AI opponent right. I wanted the minimax algorithm to work across difficulty levels, but at first, ChatGPT gave me broken or overly simplified versions. After rephrasing my request and walking through the logic together, we managed to build an AI that felt both fun and fair.

Name: Sravani Pati **Prof: Paige Anne Rodeghero**

CUID: C79908900 **CPSC 8740**



Challenges Faced While Using ChatGPT for all three applications:

Throughout the development of all three applications, one of the main challenges I faced was getting ChatGPT to fully understand what I wanted. I often had a clear idea in mind, whether it was a layout, a feature, or a specific behavior, but the responses didn't always match my expectations. Sometimes the code didn't work, other times the logic was off, or it simply missed the point.

To solve this, I learned to be more specific in how I explained things. When a conversation started to drift or get confusing, I would open a new chat and explain the entire idea again, step by step. This fresh start often helped lead to better and more accurate results.

Even though it took multiple attempts at times, ChatGPT still played an important role in every part of the project. It helped me move forward, gave structure to my ideas, and often offered suggestions I hadn't thought of. The key was that I had to lead the process. ChatGPT supported me, but I had to guide it, test everything, and make the final decisions. That collaboration helped turn my ideas into working apps.

Name: Sravani Pati **Prof: Paige Anne Rodeghero**

CUID: C79908900 **CPSC 8740**

3. Analysis and Reflection

Effectiveness of ChatGPT in Software Development

Using ChatGPT throughout this project showed me how far AI tools have come in supporting real development tasks. It was not perfect, but it was powerful.

One of the biggest strengths of ChatGPT is how quickly it can turn ideas into starting points. Instead of spending hours searching Stack Overflow, reading long documentation, or testing random snippets from the internet, I could ask ChatGPT a direct question and get a focused, often working, answer within seconds. It helped me build layouts, fix errors, understand unfamiliar libraries, and even suggested extra features that I hadn't considered before. It felt like having a personal programming assistant that I could talk to at any time.

Another huge advantage is how natural and open the interaction feels. Unlike search engines where you have to carefully phrase your query, with ChatGPT I could just speak my mind like even casually as well and still get a meaningful response. It's like talking to a teammate who is always available, never tired, and doesn't judge any question. That alone made the entire development experience smoother and more enjoyable.

However, ChatGPT is not without its limitations. Sometimes it gave wrong answers or generated code that didn't make sense. There were moments when it misunderstood what I asked, or gave suggestions that looked good but failed during execution. I often had to take a step back, explain my request more clearly, or restart the chat entirely. These moments reminded me that AI is not a replacement for understanding. It still needs direction, and it can't read your mind. If I wasn't clear, it couldn't help. And even when it was clear, the output always needed testing and refining.

That's why I see ChatGPT not as a magic tool, but as a smart collaborator. It's very effective when used thoughtfully, but it doesn't do everything for you. The real value came from combining its suggestions with my own thinking, creativity, and testing.

Reflection on Integration and the Future of Development

Looking at how ChatGPT helped me in this project, I can already see how it will be a part of my future workflow. I no longer feel stuck when I face a problem. Instead of getting frustrated or searching endlessly, I now approach challenges as conversations. I describe the issue to

Name: Sravani Pati **Prof: Paige Anne Rodeghero**

CUID: C79908900 **CPSC 8740**

ChatGPT, see what it offers, then build on top of that. It helps me explore ideas more quickly, test different paths, and find better solutions. Even when I already know how to do something, I still ask ChatGPT to see if there's a smarter or simpler way.

Comparing this experience to how I used to build software a few years ago, the difference is massive. Back then, if I got stuck or had an idea, I would have to spend hours digging through forums, trying to find something similar. Even then, the answers weren't always helpful. Now, with one tool, I can explore ideas, learn concepts, write code, fix bugs, and even plan my projects. It saves time, reduces stress, and opens the door to learning in a much more interactive way.

That said, ChatGPT can't build great software alone. It needs human direction. It can't decide what problem to solve, how the interface should feel, or what the user experience should be like. That's where human thinking still matters most. AI can help build, but it's our job to design, decide, and lead. In a way, this makes software development more collaborative than ever not just between people, but between people and AI.

Looking ahead, I believe tools like ChatGPT will become a core part of how developers work. They will speed up learning, reduce routine tasks, and help us focus more on creativity and design. I don't think AI will replace developers. Instead, it will empower them to build more, think deeper, and create better solutions.

This project gave me a clear picture of what that future looks like. With the right mindset, AI becomes more than a tool. It becomes a partner.