Working with Aider throughout this assignment has been an insightful experience that highlighted both the strengths and limitations of AI-assisted development. I found that one of the most effective techniques was using natural language prompts to guide Aider through implementing new features. By adding relevant files to the context with /add, I could ask detailed questions about how the React Calculator worked before making any changes. This helped me understand the structure of the application and ensured that I didn't break any existing logic while enhancing functionality.

When implementing features like keyboard input support and backspace functionality, I discovered that refining my initial prompts often led to better results. For example, if Aider's first attempt at adding keyboard support didn't fully integrate with the calculator's state management, I could ask for a revision by specifying what parts needed improvement. The ability to interactively review proposed code changes using /diff was also incredibly useful. It allowed me to see exactly what modifications were being made and decide whether they aligned with my goals before accepting them.

However, there were some limitations during the process. One recurring issue was related to UI styling. In one case, Aider added a calculation history panel, but used black text on a black background, making it unreadable. This showed that while Aider is excellent at generating functional logic, it sometimes lacks awareness of visual design considerations. I had to manually adjust the styles or refine my request to include specific instructions for contrast and visibility. Additionally, managing state across multiple features required careful coordination, and occasionally Aider would propose changes that conflicted with existing logic — which meant I had to intervene and revise the implementation.

Compared to traditional coding workflows, Aider significantly reduced the time spent writing repetitive logic and boilerplate code. Instead of focusing heavily on syntax and structure, I could describe what I wanted to accomplish and let Aider generate the foundational code. This allowed me to concentrate more on integration, testing, and refinement. However, unlike manual coding where you build everything step-by-step, working with Aider required a different kind of thinking — more focused on communication, feedback, and iteration. It felt less like writing code line by line and more like collaborating with a very knowledgeable assistant who could do much of the groundwork, provided I gave clear instructions and reviewed the output carefully.

To improve Aider for future use, I believe it would benefit from built-in tools for real-time preview of UI changes, better handling of style suggestions, and deeper Git integration that supports feature branching automatically. Also, having Aider provide test cases alongside implemented features would help ensure quality and reliability early in the development cycle. Finally, better error detection and suggestions for fixing common issues — such as color contrast problems or broken imports — would make the tool even more effective for developers at all levels.

Throughout this assignment, I learned the importance of prompt engineering and iterative development when working with AI tools. Aider acted as a collaborative partner rather than a replacement for my own understanding of the codebase. It excelled at scaffolding logic and explaining code structure, but still required human oversight to ensure correctness and usability.

This experience taught me that AI-assisted development is not about removing the developer from the equation, but about enhancing productivity and enabling faster exploration of ideas. With practice, I believe these tools can become essential companions in modern software development, especially when used thoughtfully and iteratively.