**Vibe Coding Experience Report**

Willy Jahner

**Introduction**

The purpose of this assignment is to experience “vibe coding”, compare different Large Language Models (LLMs) as coding and problem-solving partners, apply cryptographic reasoning to solve an Aristocrat cipher, and reflect on how LLM assistance affects one’s problem-solving process. The approach to this assignment was to break it up into three separate parts. First, I ensured that I had everything set up for the vibe coding sessions, which included generating an Aristocrat cipher text and obtaining access to ChatGPT and Google Gemini. The next part of the assignment was the actual vibe coding sessions. Finally, the last part of the assignment was for the deliverables, which includes this Experience Report, along with providing a working cipher-cracking program and reproducibility bundle.

**Approach and Tools**

My first steps for this assignment was to choose which LLMs to use for the vibe coding sessions. I use ChatGPT a lot in my everyday life, so I chose this LLM because I am quite familiar with it. I also chose to use Google Gemini as my second LLM. I was not familiar with Google Gemini because I have never used it, but I know it is a very popular tool. The version of ChatGPT I used was GPT-5. The version of Google Gemini I used was 2.5 Flash.

Guided by the directions for the assignment, I did not plan to develop the Aristocrat cipher-cracking program in any particular programming language but instead had the vibe coding session guide that decision. It turned out that my first vibe coding session with ChatGPT led me down a path of implementing the program in a Python script. Therefore, since I had already developed a Python script during my vibe coding session with ChatGPT, I specifically targeted a Python script in my vibe coding session with Google Gemini to keep things consistent. I also needed to set up a Python environment on my personal computer because I have never needed a Python environment until now. Therefore, for this assignment, I used the latest released version of Python at the time, which is 3.13.7.

**Interaction Narrative**

Due to the amount of text generated during the vibe coding sessions, I saved the links to the dialogs. My vibe coding sessions can be found here:

* ChatGPT: <https://chatgpt.com/share/68d60773-c24c-8009-9663-cbb76a084288>
* Google Gemini: <https://gemini.google.com/app/9e37102f922ae86e>

In addition, I saved Word documents with the entire dialogs captured in case these links could not be accessed. These Word documents can be found in the same repository as the final Python script.

In summary, my vibe coding sessions began with me asking the LLM to explain what an Aristocrat Cipher is. I then followed this up with explaining what I was trying to accomplish and asked if they could help me achieve it. The LLMs then began generating code for me. I would test this code, see that it failed, and work with the LLMs to refine the code to meet my needs. I continued this process until I had a working Python script.

**Comparative Insights**

When comparing ChatGPT (version GPT-5) and Google Gemini (version 2.5), I found that ChatGPT is better at generating code. In fact, it was even more eager to give me implementations of code, whereas I had to prod Google Gemini a little more for implementations. For this assignment, I found that ChatGPT gave me better quality code than Google Gemini. I was able to work with ChatGPT to find an implementation of an Aristocrat solver that was successful, but after many iterations with Google Gemini, we never came up with a successful solution. In addition, the ChatGPT solution was much simpler with about half as many lines of code. In other ad hoc research I have done, it seems that ChatGPT is a little better to use for help with implementing code.

Both ChatGPT and Google Gemini are similar in how they respond to prompts. Both are very good at summarizing information, and they also both add in extra information that may be valuable. In addition, both tools are extremely fast in gathering the necessary information and outputting it to the user.

**Reflections**

I was surprised at how fast both AI tools are able to generate code, and not only just a little code, but hundreds of lines of code in seconds. It is also surprising to me how we can have seemingly human-to-human type conversations with these AI tools. We essentially are able to talk to them and ask questions the way we would when performing pair programming. However, it is important to note that we need to double-check the implementations generated by these tools because they are not always 100% accurate and may contain mistakes or bugs. Oftentimes, they may need just little tweaks, but other times there are important areas of the code that are incorrect. In addition, if I were to ask these tools the same prompts again, I would potentially receive different implementations each time. Using these AI tools is not the best option if focused on reproducibility. With all of that said, these AI tools can be very useful for any kind of code development activity that one is embarking on.