This report documents the process of testing and refining prompts for training a large language model (LLM), including the GPT-4o, Mistral-8x22B-Intruct-v01, Claude 3.5, Gemini 1.5 Pro model, to recognize various classical cipher types, including Baconian, Autokey, Gronsfeld, Vigenere, Railfence. The goal was to create prompts that could effectively instruct the LLM to identify ciphers and calculate specific metrics. The process involved multiple iterations of prompt design, testing, and refinement to achieve the desired output format and accuracy.

Prompt Design and Testing Process

1. Initial Prompt Design:

Each cipher has 10 examples, "xxx" here is the ciphertext. I used symbols like <<<, >>>, """ to distinguish ciphertext, code from each other and ciphertext, code from instructions

```
I will give you some 10 examples for 5 ciphers Baconian, Autokey, Vigenere, Railfence, Gronsfeld. The ciphertexts are placed in <<<, >>>, they are divided with """
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Cipher Baconian:
<<<
"""
xxx
"""
xxx
"""
xxx
"""
>>>
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Cipher Autokey:
<<<
"""
xxx
"""
xxx
"""
xxx
"""
>>>
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Cipher Vigenere:
<<<
"""
xxx
"""
xxx
"""
xxx
"""
>>>

Cipher Railfence:
<<<
"""
xxx
"""
xxx
"""
xxx
"""
>>>
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Cipher Gronsfeld:
<<<
```

```
"""
xxx
"""
xxx
"""
xxx
"""
>>>
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
```

You should analyse all the ciphetexts in the examples I gave you first. Then, I will give you the ciphertext and your task is identify which type of cipher used in 5 ciphers in example and give me the explaination.
The ciphertext :
<<yctemeoivoaycoiafhngsatdcntooetfrlbairdtoeilansiesungrntowdltedvnindvoyienhttadnehmolbroioirftinmere >>>

Outcome: LLM returned results but did not focus on identifying the cipher used but rather focused on explaining how each type of encryption worked.

2. Ask more specific questions

Put this sentence at the end of the prompt

I don't want to explore how these ciphers are implemented, or are trying to decrypt these texts, just give me the final answer and the full explaination.

Outcome: LLM still explained how the ciphers work and answered which cipher used to encrypt the text, sometimes answered with 2 types of cipher. Howerver, LLM will sometimes throw an error on the Baconian cipher part, saying that my example is wrong because Baconian only uses A and B or something like that.

3. Add the python code

Add the python code at the beginning of the prompt

Here are the codes for encryption of each type of cipher, including Baconian, Autokey, Gronsfeld, Vigenere, Railfence.
<<<

```
def encrypt_Baconian(self, plaintext, key):
        ciphertext = []
        for p in plaintext:
            for k in key[p]:
                if k < 13:
                    r = random.randint(0, 12)
                else:
                    r = random.randint(13, 25)
                if r in (9, 21):  # remove j and v
                    r -= 1
                ciphertext.append(r)
        return np.array(ciphertext)


def encrypt_Autokey(self, message, key):
        cipher = []
        k_index = 0
        # here the key has to be converted back to a list as it must be extended.
        key = list(key)
        for i in message:
            text = i
            text += key[k_index]
            key.append(i)  # add current char to keystream
            text %= len(self.alphabet)
            k_index += 1
            cipher.append(text)
        return np.array(cipher)

def encrypt_Gronsfeld(self, plaintext, key):
        ciphertext = []
```

```
        for i, p in enumerate(plaintext):
            ciphertext.append((p + key[i % len(key)]) % len(self.alphabet))
        return np.array(ciphertext)

def encrypt_Vigenere(self, plaintext, key):
        key_length = len(key)
        ciphertext = []
        for position in range(len(plaintext)):
            p = plaintext[position]
            if p >= len(self.alphabet):
                ciphertext.append(self.unknown_symbol_number)
                continue
            shift = key[(position - ciphertext.count(self.unknown_symbol_number)) % key_length]
            c = (p + shift) % len(self.alphabet)
            ciphertext.append(c)
        return np.array(ciphertext)

def encrypt_Railfence(self, plaintext, key):
        ciphertext = []
        row_size = len(key[0])
        rows = [[] for _ in range(row_size)]
        pos = 0
        direction = 1
        for i in range(len(plaintext) + key[1]):
            if i >= key[1]:
                rows[pos].append(plaintext[i-key[1]])
            pos += 1 * direction
            if pos in (row_size - 1, 0):
                direction = direction * -1
        for i in range(len(rows)):
            ciphertext += rows[np.where(key[0] == i)[0][0]]
        return np.array(ciphertext)
>>>
```

Outcome: LLM realised that code might be adding randomness instead of directly mapping the letters to a binary code. Howerver, when I used the ciphertext that appears in the examples, LLM still answered incorrectly. Thus, I thought I have used the wrong format and the LLM can not fully understand the examples.

4. Refinement of prompt structure

Write example in json object

```
I will give you some examples of 5 ciphers Baconian, Autokey, Vigenere, Railfence, Gronsfeld as a Json object
{
            "Baconian" : ["xxx","xxx","xxx"],
            "Autokey" :["xxx","xxx","xxx"],
            "Vigenere" :["xxx","xxx","xxx"],
            "Railfence" :["xxx","xxx","xxx"],
            "Gronsfeld" :["xxx","xxx","xxx"],
}
```

Outcome: LLM can now answered correctly when I used the old ciphertext. But most of the answers are very random, out of 10 tries, only 3 were correct the first time, and 1 more was correct the second time after I said the first answer was wrong. Because of this, I think the instructions are not complete and good enough and LLM does not spend much time calculating the answer.
Sometimes, LLM (mostly Gemini) asked for more detailed features of the ciphertext such as Index of Coincidence, Kasiski Examination, Frequency Analysis. According to my observations, all of the LLM are unable to calculate the IoC without runing the python code. Errors when I let LLM calculate IoC include counting the number of occurrences of the wrong character, performing wrong calculations and writing wrong formulas

5. Strict output requirements

Act as a Cryptographer, I have a ciphertext that is encrypted by one of these 5 ciphers: Baconian, Autokey, Gronsfeld, Vigenere, Railfence, and you job is to analyse the code and the examples first and than identify which type of cipher used to encrypt the text.
.... {python code}
.... {examples}

You should analyse all the ciphetexts in the examples I gave you first. **Note, plain text has all spaces removed.** Than I will give you the ciphertext and your task is identify which type of cipher used in 5 ciphers in example and give me the explaination. I don't want to explore how these ciphers are implemented, or are trying to decrypt these texts, don't require more information except what I give you, just give me the final answer and the full explaination.

The ciphertext : <<<xxx>>>

Which type of cipher used to encrypt the text above and there are only 5 options: Baconian, Autokey, Gronsfeld, Vigenere, Railfence, knowing that the IoC of the cipher is xxx. Please provide a comprehensive response of at least 1000 words.

Outcome: I need to repeat the question and requirement lots of time because LLM sometimes didn't answer what I asked. LLM had a promblem in identifying the cipher Railfence and Baconian with the correct rate of 0 out of 4 tries and I think it "prefers" the other 3 ciphers. Because LLM's answer is always one of the 3 ciphers Autokey, Gronsfeld, Vigenere with a correct rate of 3 out of 6 tries (nearly random). Adding information like IoC also does not affect this.

6. Add IoC value to the example

I present the data in a table, with 3 columns separated by "|"

```
Next, I will give you some examples of 5 ciphers Baconian, Autokey, Vigenere, Railfence, Gronsfeld with their Idex of Coincidence in from of a table . Note, plain text has all spaces removed.
| Type of Cipher | Cipher | IoC |
|-----|----|-----|
|Baconian| xxx | xxx |
  . . .
|--------------|

First, you should analyze all the ciphertexts and their value IoC in the examples I gave you by comparing them with each other. Than I will give you the ciphertext and your task is identify which type of cipher used in 5 ciphers in example and give me the explaination. I don't want to explore how these ciphers are implemented, or are trying to decrypt these texts, don't require more information except what I give you, just give me the final answer and the full explaination.
```

Outcome: The LLM can understand the data in form of a table very well. I did it by using the ciphertext given in the example or asked for information about the IoC along with asking it to average the IoC of each type. Because of the difference in IoC values of Railfence cipher (0.063-0.071) and Baconian cipher (0.085-0.091) compared to the other 3 ciphers, the LLM can identify these 2 ciphers very well and not confuse it with the other 3. But also because of this, LLM has difficulty distinguishing Autokey, Gronsfeld and Vigenere ciphers.

7. Add the information about 1-Gram Frequency

```
| Type of Cipher | Cipher | IoC | 1-Gram Frequency|
|-----|----|-----|-----|
|Baconian| xxx | xxx | 0.02, 0.01, 0, 0.03 ...|
  . . .
|--------------|
```

Outcome: The LLM can understand the data in form of a table well, but once it confused IoC with the initial value of the 1-gram frequency. For this reason I find LLM cannot read tables 100% well and can (sometimes) be confused

8. Change the format of the example

```
Next, I will give you some examples of 5 ciphers Baconian, Autokey, Vigenere, Railfence, Gronsfeld with their Idex of Coincidence, 1-Gram Frequency. Note, plain text has all spaces removed.
 Type of Cipher: vigenere; Ciphertext: xxx; IoC: 0.0404040404040; 1-Gram Frequency: a - 0.0700, b - 0.0100, c - 0.0400 .... ;
  . . .
```

Outcome: The LLM can understand the data well, but it only concentrate on the IoC value and don't spend much attention on the 1-Gram Frequency. Moreover, the message was too long for some models and the LLM can not handle it all.

9. Don't put the ciphertext in the example

I decided not to let an LLM to understand and analyse the ciphertext, because when I tried to get it to generate a 1-Gram frequency table, it did it completely wrong, except for chatGPT 4o which has the ability to write python code and run that code to calculate the chatGPT.

```
Next, I will give you some examples of 5 ciphers Baconian, Autokey, Vigenere, Railfence, Gronsfeld with their Idex of Coincidence, 1-Gram Frequency.
 Type of Cipher: vigenere; IoC: 0.0404040404040; 1-Gram Frequency: a - 0.0700, b - 0.0100, c - 0.0400 .... ;
  . . .

First, you should analyze the python code and the characteristics of each ciphertext in the examples I gave you by comparing them with each other. For each ciphertext, you should analyse the Ioc average and distribution and the distribution of the frequency.
Than I will give you the ciphertext and your task is identify which type of cipher used in 5 ciphers in example and give me the explaination. I don't want to explore how these ciphers are implemented, or are trying to decrypt these texts, don't require more information except what I give you, just give me the final answer and the full explaination.
```

Outcome: the message is still too long for most of the model. The LLM could identify Vigenere and Autokey pretty well (both with 4 correct out of 5), but it could not identify Gronsfeld (1/5), 3 times mistaken for Vigenere.

10. Add information about Chi-Square

Outcome: The information about Chi-Square didn't support the calculation because the range of distribution of Vigenere (0.0281 to 0.0810) and Gronsfeld (0.0181 to 0.0774) ciphers are very large and similar and LLM is often confused between these 2 ciphers. The LLM could still identify 3 remaining ciphers well.