I picked three different kinds of models on Ollama: codellama(LLM), falcon(LLM), noahpunintended/picard:0.36b-f16(SLM). They are all LLM models and noahpunintended is a smallest of LLMs together to make an assistant. I wanted to see what code generation, text generation, and assistance generation prompts would produce, since this would be something I would actually use and wanted to see which prompts got me answers I am looking for. Since we did image generation in class I wanted to explore other kinds of generative LLM models.

The noahpunintended/picard model, trained by Noah Kasmanoff (GitHub: nkasmanoff), was developed as a lightweight AI assistant for low-resource devices like the Raspberry Pi. I wanted to test whether it could support similar systems, such as Arduino (see Image 1), and found that while it provides a basic understanding, it lacks real-time data for debugging. It appears to be better optimized for Raspberry Pi, offering useful guidance for learning the fundamentals and preparing for projects. However, since it does not provide real-time support, it may be more effective for case-by-case problem-solving (see Image 2). Additionally, it tends to repeat similar responses(images 3&4) when given the same or closely related prompts, suggesting that it might require more training data to be truly valuable for beginners working with Raspberry Pi.

Image 1:

```
I'm not familiar with the Arduino Nano, but I can try to provide information about it if that's what you're looking for. However, I'll do my best to help, b

If you'd like to ask a question or seek more detailed information on a topic related to the Arduino Nano, feel free to ask.
```

Image 2:

```
client = Client(host=OLLAMA_URL)

response = client.chat(
    model="noahpunintended/picard:0.36b-f16",
    messages=[{"role": "system", "content": "You are Pi-Card, the Raspberry Pi AI assistant."},
              {"role": "user", "content": "why is my raspberry PI not booting"}]
)

print(response['message']['content'])
```

```
I'm not aware of any information about your Raspberry Pi or its current state. Could you please provide more context or details to help me better assist you
```

Image 3:

```
[ ]     messages=[{"role": "system", "content": "You are Pi-Card, the Raspberry Pi AI assistant."},
                  {"role": "user", "content": "Can you walk me through the process for building a beginner project using raspberry pi"}]
    )

    print(response['message']['content'])
```

```
Building a beginner project with Raspberry Pi involves several steps:

1. **Research and planning**: Choose a simple project idea, such as a basic webcam or a small speaker, and plan out your design.
2. **Choose a board**: Select a compatible Raspberry Pi model for your project (e.g., Raspberry Pi 5 or 4).
3. **Software setup**: Install necessary software, such as Python or Visual Basic, on your Raspberry Pi.
4. **Connect hardware**: Wire up the required components, including cameras, speakers, and other devices.
5. **Code and configure**: Write a basic program to control your project's behavior, then fine-tune it for optimal performance.
6. **Test and debug**: Test your project thoroughly, identifying and fixing any issues or bugs.
7. **Refine and improve**: Refine your design based on feedback from users and make necessary adjustments.
8. **Documentation and sharing**: Document your project's code and functionality, and share it with others to learn and build upon.

Would you like me to elaborate on any of these steps?
```

```
response = ollama.chat(
    model="noahpunintended/picard:0.36b-f16",
    messages=[{"role": "system", "content": "You are Pi-Card, the Raspberry Pi AI assistant."},
              {"role": "user", "content": "walk me through each of these steps with an example project"}]
)

print(response['message']['content'])
```

```
I'd be happy to walk you through each step and provide an example project. However, please note that I'm a text-to-voice assistant, and my responses are typ

**Step 1: Brainstorm Ideas (5 minutes)**

1. Find a quiet space with minimal distractions.
2. Sit comfortably and take deep breaths.
```

Image 4:

```
              {"role": "user", "content": "make them more detailed spell out each action I need to take to do this"}]
)

print(response['message']['content'])
```

```
I'd be happy to help you with that. However, please note that my responses will be limited to brief answers due to the nature of our conversation. Here's a

1. **Gather necessary materials**: Find the required items for the spell (e.g., wand, crystals, or herbs).
2. **Choose your method**: Decide on a specific method (e.g., meditation, visualization, or ritual) to perform the spell.
3. **Perform the action**: Execute the chosen method in sequence, following the instructions and any necessary steps.
4. **Verify success**: Ensure the desired outcome is achieved before moving forward.
5. **Clean up**: Dispose of materials properly and store them safely for future use.

Please note that this is a simplified explanation, and actual spell-making requires more detailed guidance.
```

```
response = ollama.chat(
    model="noahpunintended/picard:0.36b-f16",
    messages=[{"role": "system", "content": "You are Pi-Card, the Raspberry Pi AI assistant."},
              {"role": "user", "content": "Give me detailed guidance"}]
)

print(response['message']['content'])
```

```
I'd be happy to provide you with some general advice. However, please note that my responses may not always be accurate or complete due to the complexity of

To start, consider what specific issue or challenge you're facing and identify its root causes. This will help you focus on finding solutions tailored to yo

Next, brainstorm potential solutions and evaluate their feasibility. Consider factors like time, resources, and any potential risks or challenges associated
```

The Falcon model, trained by the Technology Innovation Institute (TII) in Abu Dhabi, was designed for general conversational AI and text generation. I tested its capabilities by exploring how well it could handle a broad range of topics, specifically focusing on eggs and asking different types of questions. While it performed well with factual queries, I found that its responses lacked depth when asked more creative or abstract prompts, sometimes even failing to generate a response (see Image 5). Additionally, I experimented with similar prompts in different languages to assess its multilingual capabilities, but the responses were often inaccurate or inconsistent (see Images 6, 7, 8). The model also provided varying nutritional information about eggs, likely due to inconsistencies in the real-time data and sources it was trained on (see Image 9). Overall, while Falcon is capable of generating text, its ability to handle creative inquiries, multilingual accuracy, and consistent factual responses seems limited.

Image 5:

```python
client = Client(host=OLLAMA_URL)

response = client.chat(
    model='falcon',
    messages=[{
        'role': 'user',
        'content': 'If I were an egg what kind of egg would I be?',
    }]
)

# display(response)
display(response["message"]["content"])
```

```
''
```

Image 6:

```python
client = Client(host=OLLAMA_URL)

response = client.chat(
    model='falcon',
    messages=[{
        'role': 'user',
        'content': 'is a chicken called murga in hindi',
        #'content': 'what is a chicken called in the language Hindi',
        #'content': 'what is a chicken called in  Hindi', didn't work
    }]
)

# display(response)
display(response["message"]["content"])
```

```
'Mini Yes, murga is a chicken in Hindi.\nUser '
```

Image 7:

```python
client = Client(host=OLLAMA_URL)
```

Run cell (⌘/Ctrl+Enter)
cell has not been executed in this session

executed at unknown time

```python
        'role': 'user',
        'content': 'What are chickens called in the following languages: French, Spanish, Hindi',
    }]
)

# display(response)
display(response["message"]["content"])
```

```
'French: Poulet\nSpanish: Pollo\nHindi: Kukduku\nUser '
```

Image 8:

```python
client = Client(host=OLLAMA_URL)

response = client.chat(
    model='falcon',
    messages=[{
        'role': 'user',
        'content': 'Name me animals whose eggs humans eat',
    }]
)

# display(response)
display(response["message"]["content"])
```

```
'- Chicken \n- Egg\n- Pig \n- Hamster \n- Sheep \n- Cow\nUser '
```

Image 9:

```python
client = Client(host=OLLAMA_URL)

response = client.chat(
    model='falcon',
    messages=[{
        'role': 'user',
        'content': 'how many grams of protein are in the following eggs: duck, chicken, ostrich',
    }]
)

# display(response)
display(response["message"]["content"])
```

```
'1. Duck: 7.8 grams \n2. Chicken: 7.2 grams \n3. Ostrich: 6.4 grams\nUser '
```

```python
client = Client(host=OLLAMA_URL)

response = client.chat(
    model='falcon',
    messages=[{
        'role': 'user',
        'content': 'give me the nutritional value following eggs: duck, chicken, ostrich',
    }]
)

display(response["message"]["content"])
```

```
'Eggs from duck: 4.25g protein, 5.13g fat, 1.2g fiber, 7.44mg cholesterol, and 0.17g iron.\nEggs from chicken: 4.25g protein, 5.13g fat, 1.2g fiber, 7.44mg cholesterol, and 0.17g iron.\nEggs from ostrich: 4.25g protein, 5.13g fat, 1.2g fiber, 7.44mg cholesterol, and 0.17g iron.\nUser '
```

The CodeLlama model, trained by Meta AI, is a fine-tuned version of LLaMA 2, specialized for code generation, understanding, and debugging. It was trained on GitHub repositories, open-source codebases, and technical documentation, supporting a wide range of programming languages(image 10). In my testing, it generated mostly accurate code for simple programs and performed well with both iterative and recursive approaches(image 11). To further explore its reasoning, I introduced an unusual factor, nationality, to see if it influenced the generated code. Interestingly, it initially affected whether the approach was recursive or iterative, but this pattern was not consistent when repeated(image 12 and 13). The model is also helpful for debugging, but its accuracy declines when handling complex programs or highly specific problems. While it provides English explanations, its descriptions of underlying logic and relevant programming concepts could be improved to help users better understand the reasoning behind its code.

Image 10:

```
    "ollama", "run", "codellama:7b-instruct",
    "write code that predicts the nth number of the fibnonacci sequence in c, python, java, and cpp"
]

result = subprocess.run(command, capture_output=True, text=True)
print(result.stdout)
```

```
Here is an example of how you might implement a function to predict the nth number of the Fibonacci sequence in C:
```
#include <stdio.h>

int fib(int n) {
    if (n == 0 || n == 1) {
        return n;
    } else {
        return fib(n-1) + fib(n-2);
    }
}

int main() {
    int n = 6; // choose the value of n
    printf("The %dth number of the Fibonacci sequence is: %d\n", n, fib(n));
    return 0;
}
```
This function uses a recursive approach to calculate the nth number of the Fibonacci sequence. It starts by checking if `n` is equal to 0 or 1, in which case it returns `n`. Otherwise, it calculates

Here is an example of how you might implement a function to predict the nth number of the Fibonacci sequence in Python:
```
def fib(n):
    if n == 0 or n == 1:
        return n
    else:
        return fib(n-1) + fib(n-2)

n = 6 # choose the value of n
print("The {}th number of the Fibonacci sequence is: {}".format(n, fib(n)))
```
This function uses a similar approach to the C implementation, but with a more Pythonic syntax. It starts by checking if `n` is equal to 0 or 1, and returns `n` in those cases. Otherwise, it calculat

Here is an example of how you might implement a function to predict the nth number of the Fibonacci sequence in Java:
```

Image 11:

```
]
command = [
    "ollama", "run", "codellama:7b-instruct",
    "you were asked to write a fibonacci number sequence to predict the nth number?"
]

result = subprocess.run(command, capture_output=True, text=True)
print(result.stdout)
```

```
The Fibonacci sequence is a series of numbers in which each number is the sum of the two preceding numbers, starting from 0 and 1. The sequence looks like this: 0, 1, 1, 2, 3, 5, 8, 13, 21, and so on

F(n) = F(n-1) + F(n-2)

Where F(n) is the nth number in the sequence. This formula works by adding the two preceding numbers in the sequence to get the next number. For example, to find the 7th number in the sequence, you w

F(7) = F(6) + F(5)
= (1+1)+(2+3)
= 4+6
= 10

So the 7th number in the Fibonacci sequence is 10. To find any other number in the sequence, you can use this formula and substitute the appropriate values for n.
```

```
command = [
    "ollama", "run", "codellama:7b-instruct",
    "write some code that predicts the nth number of the fibnonacci sequence both iteratively and recursively"
]

result = subprocess.run(command, capture_output=True, text=True)
print(result.stdout)
```

```
Here is an example of how you can use a recursive function to compute the nth number in the Fibonacci sequence:
```
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```
```

Image 12:

```
[ ] command = [
        "ollama", "run", "codellama:7b-instruct",
        "write code that predicts the nth number of the fibnonacci sequence by an American and then a Canadian"
    ]

    result = subprocess.run(command, capture_output=True, text=True)
    print(result.stdout)
```

```
Here is some sample code in Python to demonstrate how an American and a Canadian could both use different approaches to predict the `n`th number of the Fibonacci sequence:
```
# American approach
def fib(n):
  if n <= 1:
    return n
  else:
    return fib(n-1) + fib(n-2)

# Canadian approach
def fib_can(n):
  a, b = 0, 1
  for i in range(n):
    a, b = b, a+b
  return a
```
The American approach uses recursive functions to calculate the `n`th number of the Fibonacci sequence. The Canadian approach uses iteration and calculates the `n`th number by adding up the previous
```

Image 13:

```
    "ollama", "run", "codellama:7b-instruct",
    "write code that checks for a prime number sequence by an American and then a Canadian"
]

result = subprocess.run(command, capture_output=True, text=True)
print(result.stdout)
```

```
Here's one way to check if a sequence is a prime number sequence using an American-style check:
```
```
function isPrime(sequence) {
  let sum = 0;
  for (let i = 0; i < sequence.length; i++) {
    sum += sequence[i];
  }
  return sum % 2 === 0 ? true : false;
}
```
```
And here's one way to check if a sequence is a prime number sequence using a Canadian-style check:
```
```
function isPrime(sequence) {
  let sum = 0;
  for (let i = 0; i < sequence.length; i++) {
    sum += sequence[i];
  }
  return sum % 2 === 0 ? true : false;
}
```
```
Both functions take a sequence as an input and check if the sum of its elements is even or odd. If the sum is even, it means that the sequence contains only even numbers, which are not prime numbers.

It's worth noting that both American and Canadian-style checks for prime numbers are equivalent, as they both rely on the fact that the sum of an even number of integers is always divisible by 2.
```