EXP 3: Evaluation of Prompting Tools Across Diverse AI Platforms: ChatGPT, Claude, Bard, Cohere Command, and Meta

Aim:

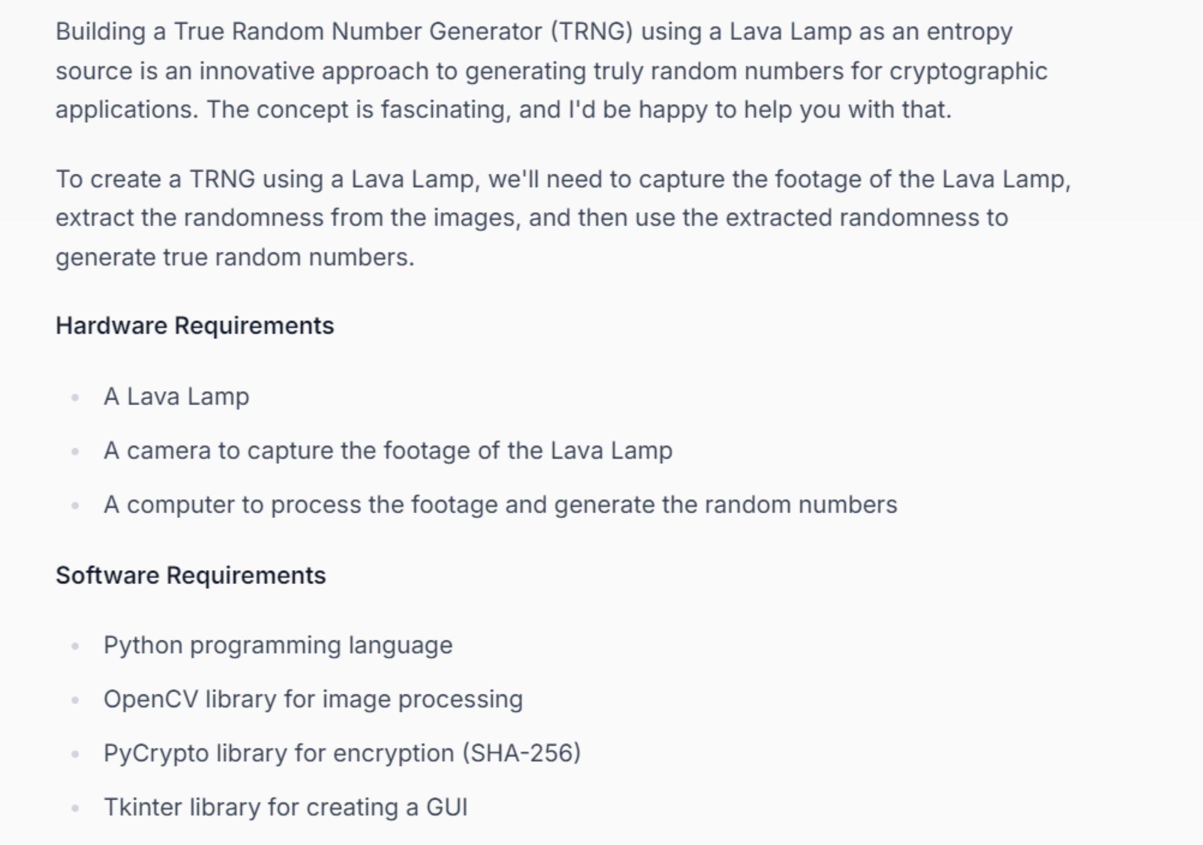
To compare the performance, user experience, and response quality of different AI platforms (ChatGPT, Claude, Bard, Cohere Command, and Meta) within a specific use case, such as summarizing text or answering technical questions.

Scenario: True Random Number Generator using Lava Lamp

Prompt:

"You are expert in cryptography; can you help me build a TRNG using Lava Lamp as entrophy source that will help me to generate true random numbers for cryptographic applications. Here are the details you need to consider Lava Lamp: Footage of Lava Lamp, Encryption: SHA-256 Include Python GUI to showcase the random numbers that are generated every 5 seconds."

Blackbox.ai [https://www.blackbox.ai] :



Building a True Random Number Generator (TRNG) using a Lava Lamp as an entropy source is an innovative approach to generating truly random numbers for cryptographic applications. The concept is fascinating, and I'd be happy to help you with that.

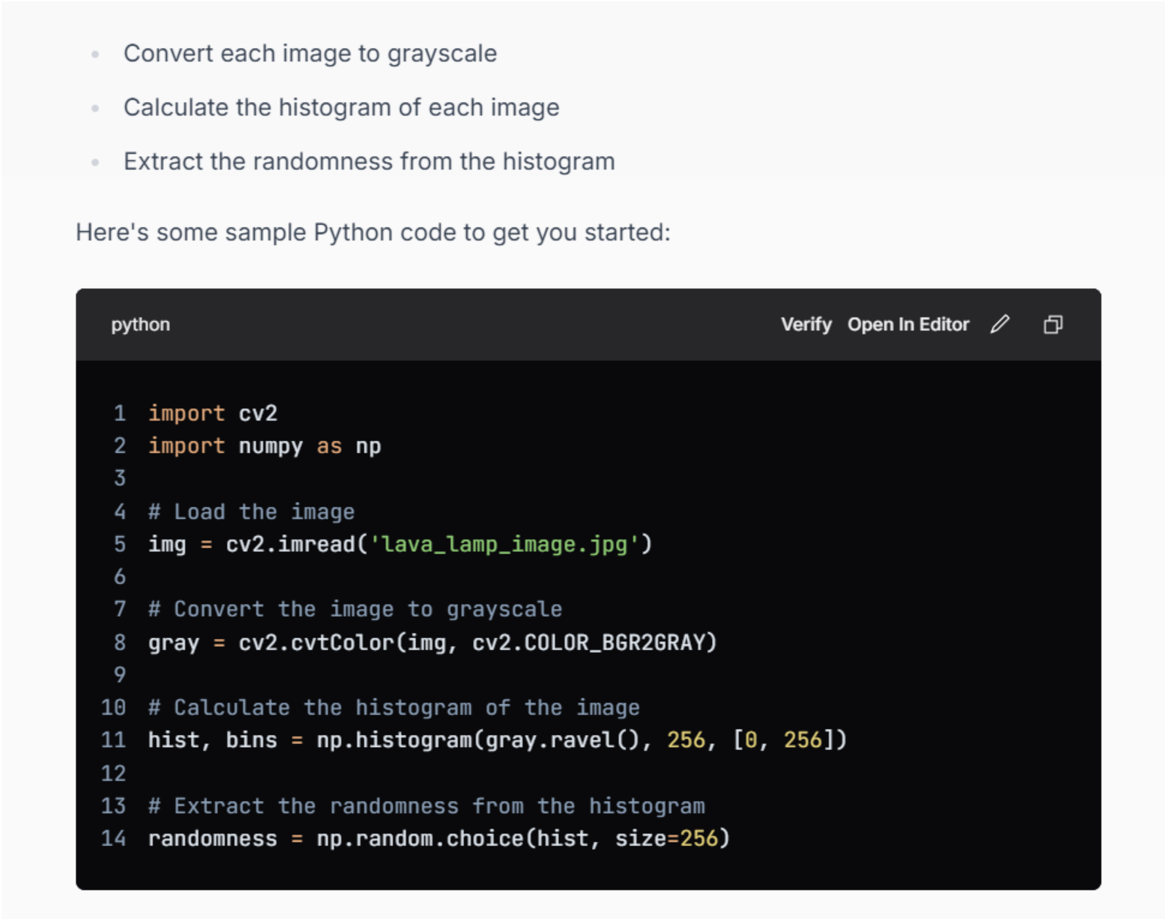
To create a TRNG using a Lava Lamp, we'll need to capture the footage of the Lava Lamp, extract the randomness from the images, and then use the extracted randomness to generate true random numbers.

Hardware Requirements

- A Lava Lamp
- A camera to capture the footage of the Lava Lamp
- A computer to process the footage and generate the random numbers

Software Requirements

- Python programming language
- OpenCV library for image processing
- PyCrypto library for encryption (SHA-256)
- Tkinter library for creating a GUI

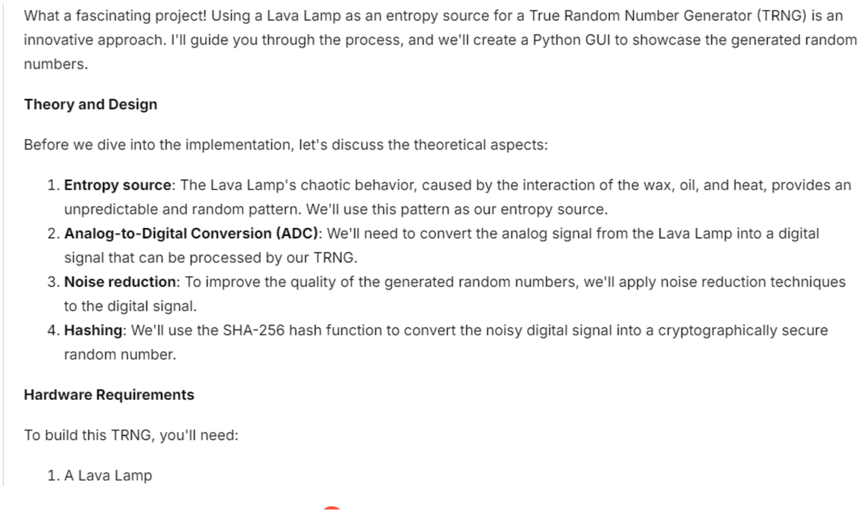


- Convert each image to grayscale
- Calculate the histogram of each image
- Extract the randomness from the histogram

Here's some sample Python code to get you started:

```python
import cv2
import numpy as np

# Load the image
img = cv2.imread('lava_lamp_image.jpg')

# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Calculate the histogram of the image
hist, bins = np.histogram(gray.ravel(), 256, [0, 256])

# Extract the randomness from the histogram
randomness = np.random.choice(hist, size=256)
```

Groq: [https://chat.groq.com/]



What a fascinating project! Using a Lava Lamp as an entropy source for a True Random Number Generator (TRNG) is an innovative approach. I'll guide you through the process, and we'll create a Python GUI to showcase the generated random numbers.

Theory and Design

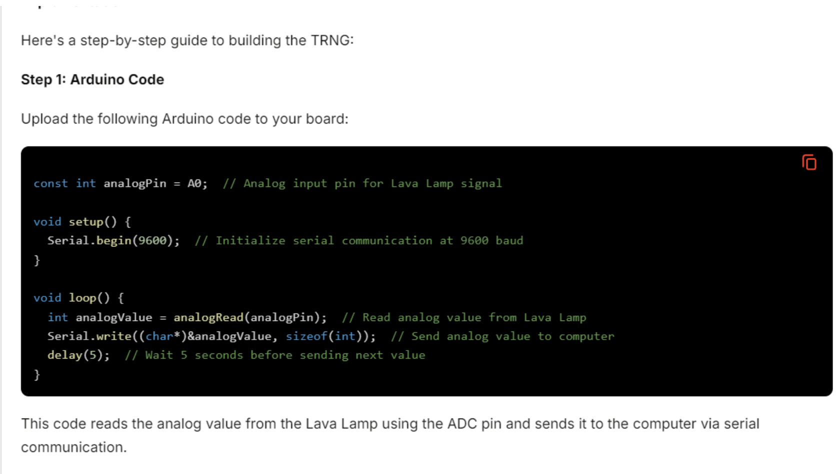
Before we dive into the implementation, let's discuss the theoretical aspects:

1. **Entropy source**: The Lava Lamp's chaotic behavior, caused by the interaction of the wax, oil, and heat, provides an unpredictable and random pattern. We'll use this pattern as our entropy source.
2. **Analog-to-Digital Conversion (ADC)**: We'll need to convert the analog signal from the Lava Lamp into a digital signal that can be processed by our TRNG.
3. **Noise reduction**: To improve the quality of the generated random numbers, we'll apply noise reduction techniques to the digital signal.
4. **Hashing**: We'll use the SHA-256 hash function to convert the noisy digital signal into a cryptographically secure random number.

Hardware Requirements

To build this TRNG, you'll need:

1. A Lava Lamp



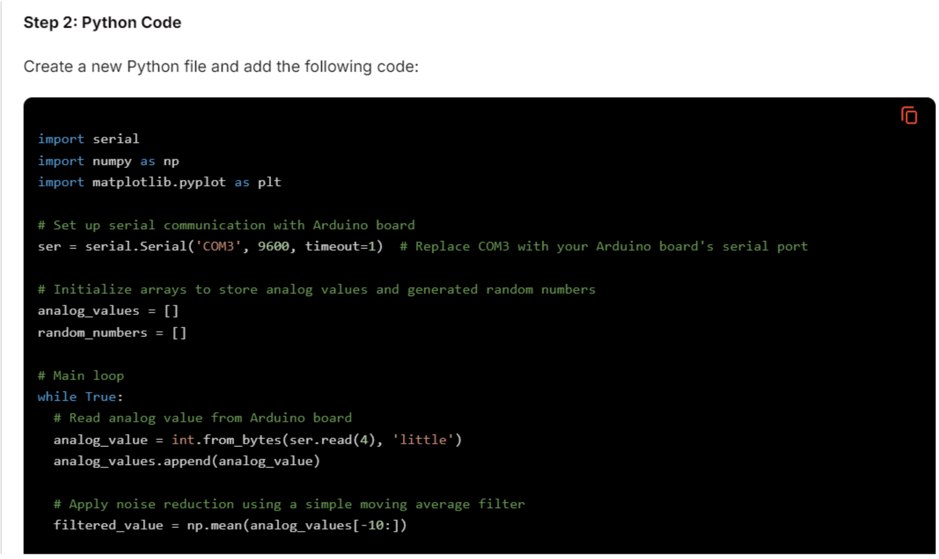
Here's a step-by-step guide to building the TRNG:

Step 1: Arduino Code

Upload the following Arduino code to your board:

```
const int analogPin = A0;  // Analog input pin for Lava Lamp signal

void setup() {
  Serial.begin(9600);  // Initialize serial communication at 9600 baud
}

void loop() {
  int analogValue = analogRead(analogPin);  // Read analog value from Lava Lamp
  Serial.write((char*)&analogValue, sizeof(int));  // Send analog value to computer
  delay(5);  // Wait 5 seconds before sending next value
}
```

This code reads the analog value from the Lava Lamp using the ADC pin and sends it to the computer via serial communication.

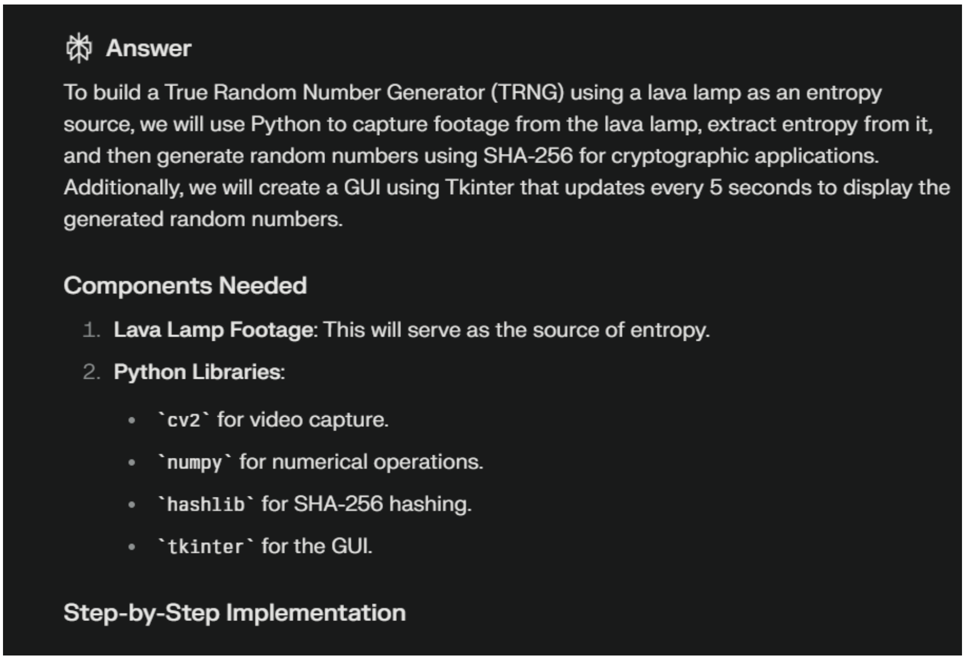


Step 2: Python Code

Create a new Python file and add the following code:

```
import serial
import numpy as np
import matplotlib.pyplot as plt

# Set up serial communication with Arduino board
ser = serial.Serial('COM3', 9600, timeout=1)  # Replace COM3 with your Arduino board's serial port

# Initialize arrays to store analog values and generated random numbers
analog_values = []
random_numbers = []

# Main loop
while True:
  # Read analog value from Arduino board
  analog_value = int.from_bytes(ser.read(4), 'little')
  analog_values.append(analog_value)

  # Apply noise reduction using a simple moving average filter
  filtered_value = np.mean(analog_values[-10:])
```

Perplexity : [https://www.perplexity.ai/]



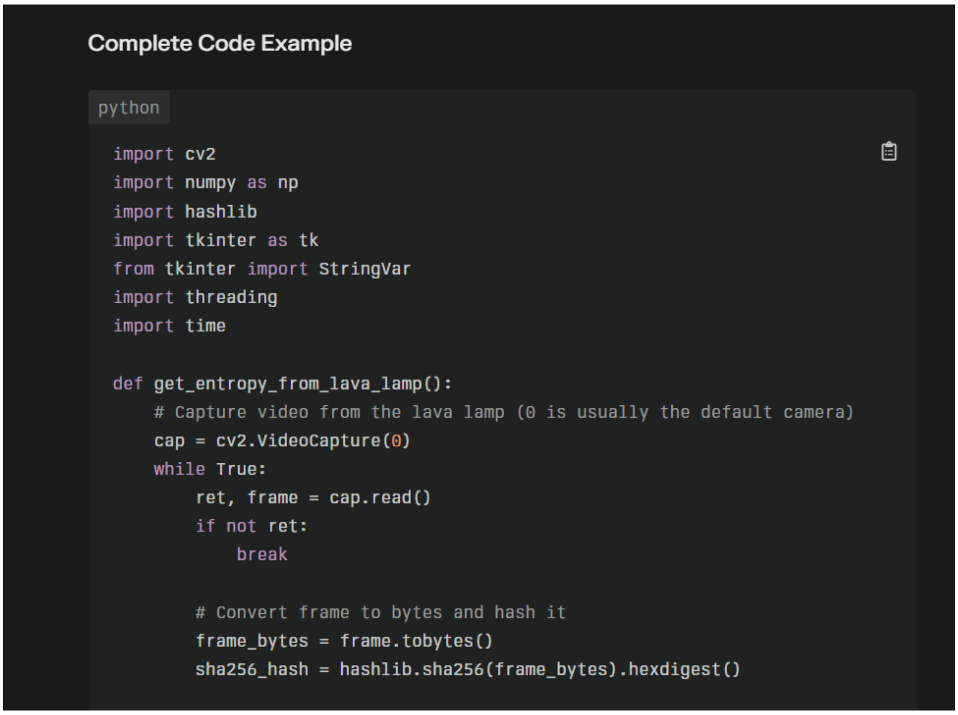
Answer

To build a True Random Number Generator (TRNG) using a lava lamp as an entropy source, we will use Python to capture footage from the lava lamp, extract entropy from it, and then generate random numbers using SHA-256 for cryptographic applications. Additionally, we will create a GUI using Tkinter that updates every 5 seconds to display the generated random numbers.

Components Needed

1. **Lava Lamp Footage**: This will serve as the source of entropy.
2. **Python Libraries**:
    - `cv2` for video capture.
    - `numpy` for numerical operations.
    - `hashlib` for SHA-256 hashing.
    - `tkinter` for the GUI.

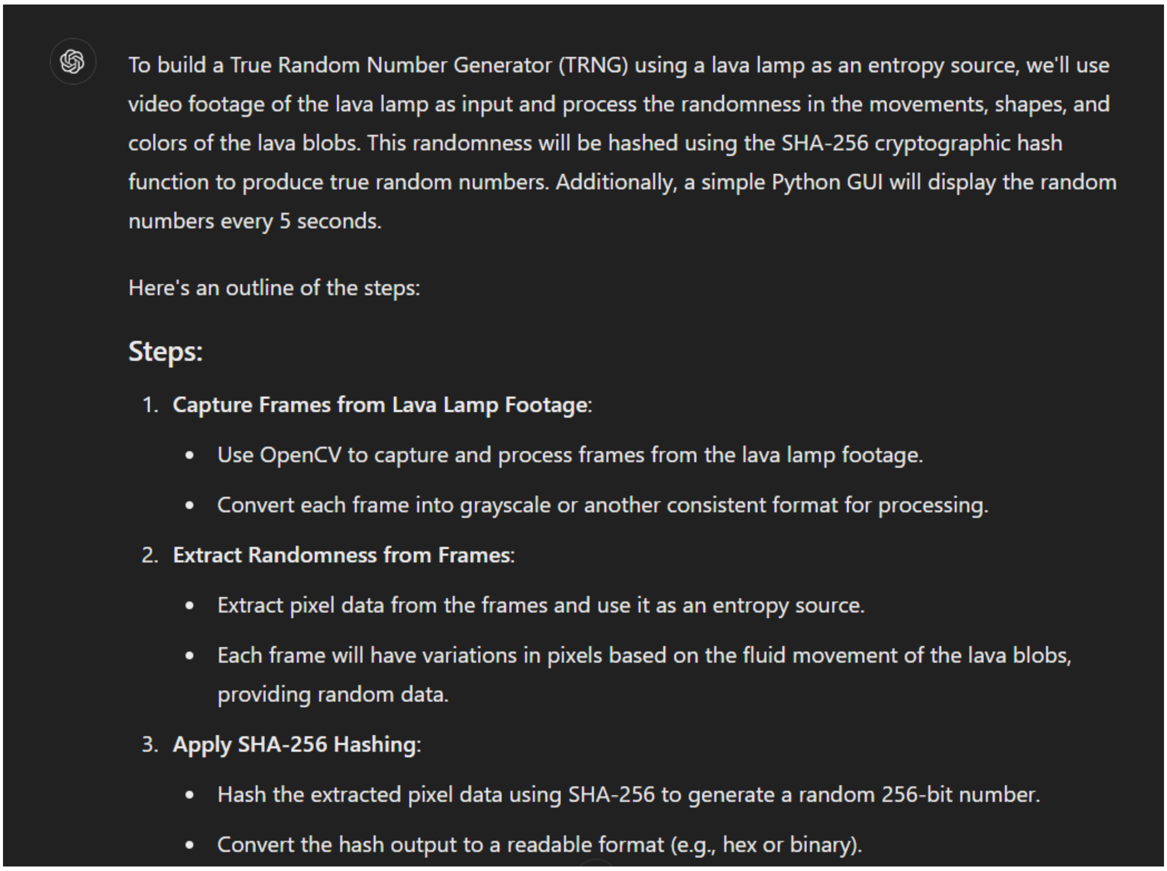
Step-by-Step Implementation



Complete Code Example

```python
import cv2
import numpy as np
import hashlib
import tkinter as tk
from tkinter import StringVar
import threading
import time

def get_entropy_from_lava_lamp():
    # Capture video from the lava lamp (0 is usually the default camera)
    cap = cv2.VideoCapture(0)
    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Convert frame to bytes and hash it
        frame_bytes = frame.tobytes()
        sha256_hash = hashlib.sha256(frame_bytes).hexdigest()
```

ChatGPT [https://chatgpt.com/]

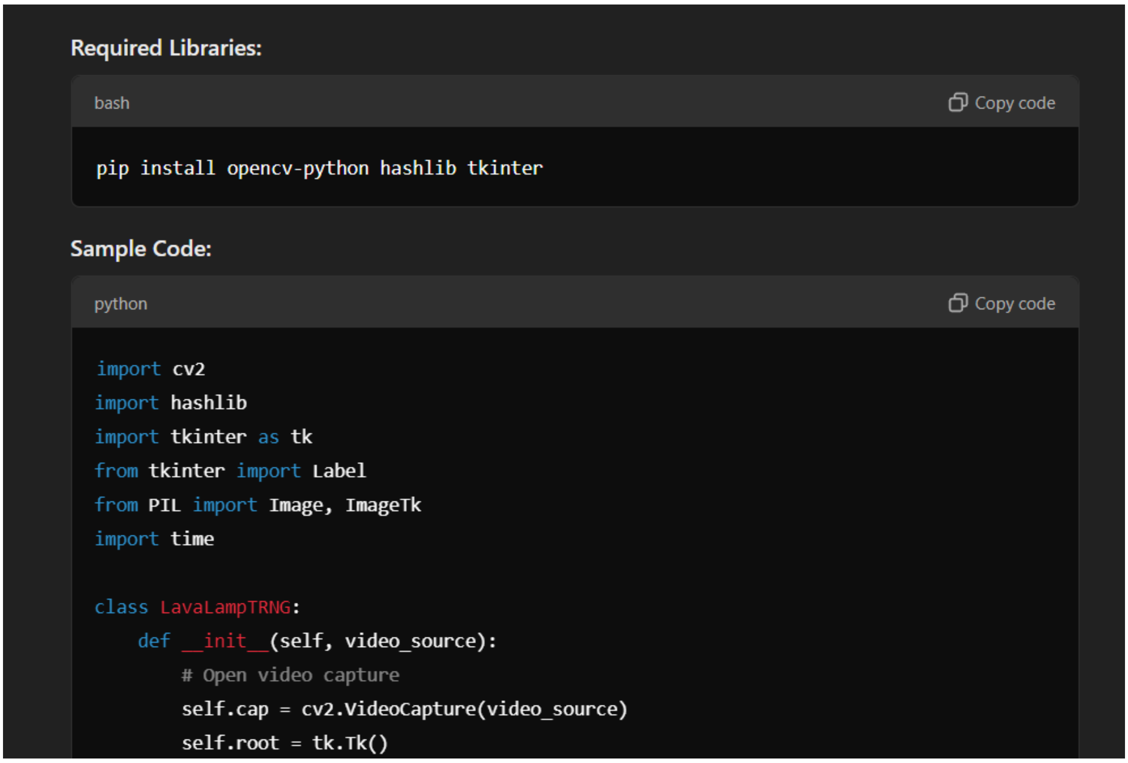


To build a True Random Number Generator (TRNG) using a lava lamp as an entropy source, we'll use video footage of the lava lamp as input and process the randomness in the movements, shapes, and colors of the lava blobs. This randomness will be hashed using the SHA-256 cryptographic hash function to produce true random numbers. Additionally, a simple Python GUI will display the random numbers every 5 seconds.

Here's an outline of the steps:

Steps:

1. **Capture Frames from Lava Lamp Footage**:
    - Use OpenCV to capture and process frames from the lava lamp footage.
    - Convert each frame into grayscale or another consistent format for processing.
2. **Extract Randomness from Frames**:
    - Extract pixel data from the frames and use it as an entropy source.
    - Each frame will have variations in pixels based on the fluid movement of the lava blobs, providing random data.
3. **Apply SHA-256 Hashing**:
    - Hash the extracted pixel data using SHA-256 to generate a random 256-bit number.
    - Convert the hash output to a readable format (e.g., hex or binary).



Required Libraries:

```bash
pip install opencv-python hashlib tkinter
```

Sample Code:

```python
import cv2
import hashlib
import tkinter as tk
from tkinter import Label
from PIL import Image, ImageTk
import time

class LavaLampTRNG:
    def __init__(self, video_source):
        # Open video capture
        self.cap = cv2.VideoCapture(video_source)
        self.root = tk.Tk()
```

Platform Accuracy Clarity Depth Relevance

GROQ	Medium: While the response accurately outlines a structured approach, some details on implementation specifics, such as handling edge cases, are missing.	High: The response is well-organized and easy to follow, using headings and bullet points to clearly delineate different aspects of the solution.	Medium: The overview is comprehensive, but it lacks deeper insights into error handling, performance optimization, and specific implementation details that may be beneficial for developers.	High: The information provided is directly applicable to the project, addressing all key requirements such as real-time data updates and modular CSS.
Perplexity	High: The response provides a clear and accurate guide for setting up a live dashboard, including component design and real-time data handling with practical code examples.	High: The structured format with clear headings and code snippets enhances readability and understanding, making it easy to follow the implementation steps.	Medium: The response offers good guidance but could include more advanced topics like state management and testing, which are crucial for scalable applications.	High: The detailed instructions and components directly align with the project requirements, ensuring relevance for building the application.

Blackbox AI	Medium: The response gives a basic overview but lacks detailed guidance on the server-side logic required for a fully functional application.	Medium: The explanation is straightforward, but some areas lack depth, making it harder for less experienced developers to fully grasp the implementation.	Low: While the response covers essential components and WebSocket integration, it does not provide enough detail on advanced features or error handling strategies.	Medium: The information is relevant but does not address all necessary components for a comprehensive solution, particularly in server implementation.
ChatGPT	High: The response provides a detailed step-by-step guide for building a React-based live dashboard, including setup, components, styling, and server-side integration for real-time updates.	High: The response is very clear and well-structured, using numbered steps and code examples to enhance understanding and usability for developers.	High: The depth is significant, covering everything from project setup to styling, real-time updates, and even a basic server implementation, offering a comprehensive framework for building the application.	High: The content is highly relevant to the project, addressing all aspects necessary for creating a functional dashboard with real-time capabilities.

Summary & Conclusion

In this comparison, we analysed the responses from four different platforms regarding the development of a React-based live dashboard for visualizing patient vitals. Each platform provided valuable insights, but they differed significantly in accuracy, clarity, depth, and relevance.

1. **GROQ** offered a solid structure but lacked some details that could enhance the implementation process, particularly around edge cases and error handling.
2. **Perplexity** provided a clear guide but could have delved deeper into advanced topics like state management and testing for scalable applications.
3. **Blackbox AI** presented a basic overview, which was less comprehensive and lacked the necessary detail for server-side implementation, making it less useful for beginners.
4. **ChatGPT** stood out with a thorough, well-structured guide that effectively covered all critical aspects of building the application, making it the most useful response overall.

Overall, the analysis shows that while all platforms offered useful guidance, the depth and clarity of the information varied significantly, impacting their practical utility for developers looking to implement a robust solution. ChatGPT's response proved to be the most comprehensive and directly applicable to the project needs, making it the top choice for developers seeking detailed instructions for building a patient vitals dashboard.