[(n) Step-1: Designing a custom PRNG

& my_prng.py

```
class MyPRNG
def __init__(self, seed: int = 123456789):
self.state = seed
self.a = 1664525
self.c = 1013904223
self.m = 2**32
def next(self) -> float:
self.state = (self.a * self.state + self.c) % self
m.
x = self.state
x ^= (x >> 11)
x ^= (x << 7) & 0x9D2C5680
x ^= (x << 15) & 0xEFC60000
x ^= (x >> 18)
return (x % self.m) / self.m
def generate_n(self, n: int):
```

return [self.next() for _ in range(n)]

Step:2 Analyze the PRNG-Uniformity

Dependency:

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from my-prng import MyPRNG
def analyze_prng():
    prng = MyPRNG(seed=42)
    nums = prng.generate(10000)
    plt.figure(figsize=(10,4))
    sns.histplot(nums, bins=50, kde=True,
    stat="density")
    plt.title("PRNG Output Distribution (uniformity check)")
    plt.ylabel("Density")
```

return [self.next() for _ in range(n)]

Step:2 Analyze the PRNG-Uniformity

Dependency:

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from my_prng import MyPRNG
def analyze_prng():
    prng = MyPRNG(seed=42)
    nums = prng.generate(10000)
    plt.figure(figsize=(10,4))
    sns.histplot(nums, bins=50, kde=True,
    stat="density")
    plt.title("PRNG Output Distribution (uniformity check)")
    plt.ylabel("Density")
```

```
plt.savefig("uniformity.png")
plt.show()
plt.figure(figsize=(6,6))
plt.plot(nums[:-1], nums[1:],
alpha=0.5)
plt.title("Lag plot (dependency check
plt.xlabel("xn")
plt.savefig("dependency.png")
plt.show()
if __name__ == "__main__":
analyze_prng()
```

Step-3: create the Github Repo structure

```
mkdir my-prng
cd my-prng
touch my_prng.py analyze_prng.py
README.md requirements.txt
```

Step 4: README.md content

My custom pseudorandom number Generator (PRNG)

This project implements a custom PRNG or an improved linear congruential generator (LCG) with added XOR bit manipulation for better diffusion and randomness.

- Reproducible seed-based PRNG
- Generates float values in [0, 1]
- Bit-mixed post-processing to improve randomness.
- Analysis via histogram and lag plot

PRNG Design Notes:

The generator uses the following:

- LCG formula: $x_{n+1} = (ax_n + c) \bmod m$

- Bit mixing : XOR - shifts for added entropy (similar to MT19937)
- Normalization : output is scaled to [0,1]

Parameters used:

- a = 1664525
- c = 1013904223
- m = 2^32