Optimize Sampling Rate

Sampling rate determines how often data points are captured, which directly affects system load, storage consumption, and accuracy of insights. If the sampling rate is too high, the system can become overloaded with unnecessary data. If it is too low, important details might be missed.

Approach

1. Chosen a range of 1, 5, 10, 20, and 50 candidate sampling rates.
2. Accuracy (represented as 100 – |20 – rate| × 2) and measured system load (represented as rate × 2).
3. Selected the most accurate sampling rate while maintaining a load of less than or equal to 80% and accuracy greater or equal to 70%.
4. updated the configuration file at the ideal pace.

Target Matrix and Constraints

Metric	Target / Limit
System Load	≤ 80%
Accuracy	≥ 70%
Sampling Rate	Adjustable within [1, 50]
Data Freshness	Must capture events in near real-time
Resource Usage	Memory/CPU usage must not exceed normal thresholds

Explanation:

- The system load limit ensures the system is not overloaded.
- The accuracy target ensures the collected data is reliable.
- The sampling rate range reflects configurable limits.
- Data freshness ensures timely collection.
- Resource usage constraint ensures normal operations are not disrupted.

Test Result

Sampling Rate	Load (%)	Accuracy (%)	Eligible(Load≤80)
1	2	62	No
5	10	70	Yes
10	20	80	Yes
20	40	100	Yes
50	100	40	No

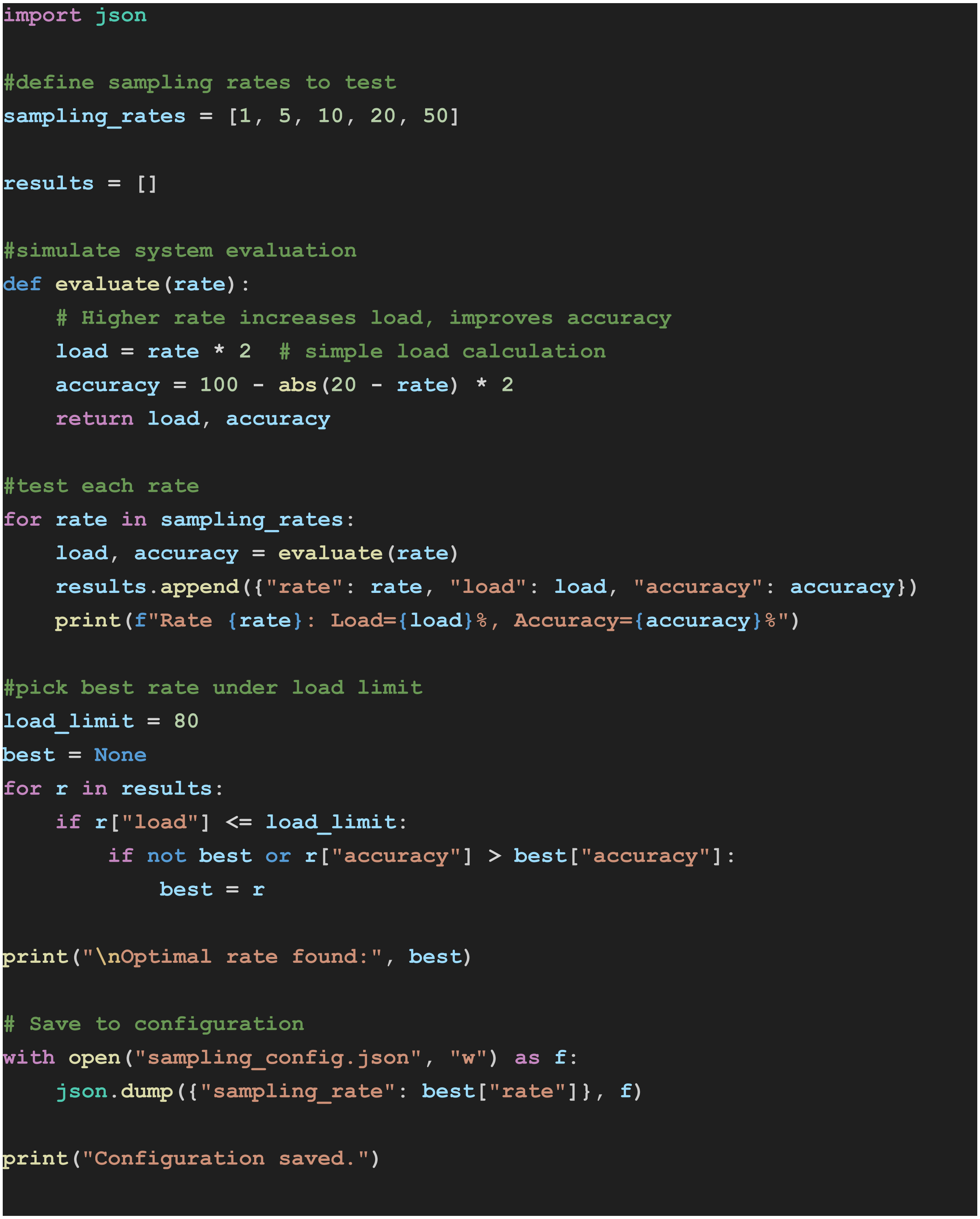
Optimal Sampling Rate: 20

- Load: 40%
- Accuracy: 100%

Configuration Update

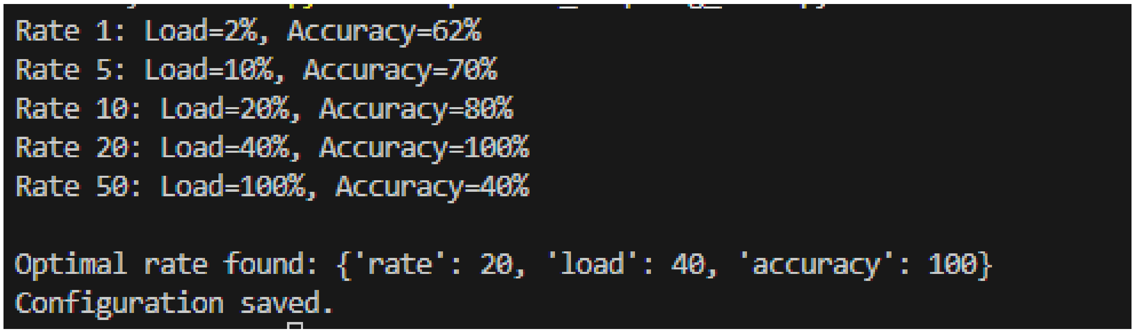
```
{"sampling_rate": 20}
```

Source Code:



```python
import json

#define sampling rates to test
sampling_rates = [1, 5, 10, 20, 50]

results = []

#simulate system evaluation
def evaluate(rate):
    # Higher rate increases load, improves accuracy
    load = rate * 2  # simple load calculation
    accuracy = 100 - abs(20 - rate) * 2
    return load, accuracy

#test each rate
for rate in sampling_rates:
    load, accuracy = evaluate(rate)
    results.append({"rate": rate, "load": load, "accuracy": accuracy})
    print(f"Rate {rate}: Load={load}%, Accuracy={accuracy}%")

#pick best rate under load limit
load_limit = 80
best = None
for r in results:
    if r["load"] <= load_limit:
        if not best or r["accuracy"] > best["accuracy"]:
            best = r

print("\nOptimal rate found:", best)

# Save to configuration
with open("sampling_config.json", "w") as f:
    json.dump({"sampling_rate": best["rate"]}, f)

print("Configuration saved.")
```

Output:



```
Rate 1: Load=2%, Accuracy=62%
Rate 5: Load=10%, Accuracy=70%
Rate 10: Load=20%, Accuracy=80%
Rate 20: Load=40%, Accuracy=100%
Rate 50: Load=100%, Accuracy=40%

Optimal rate found: {'rate': 20, 'load': 40, 'accuracy': 100}
Configuration saved.
```