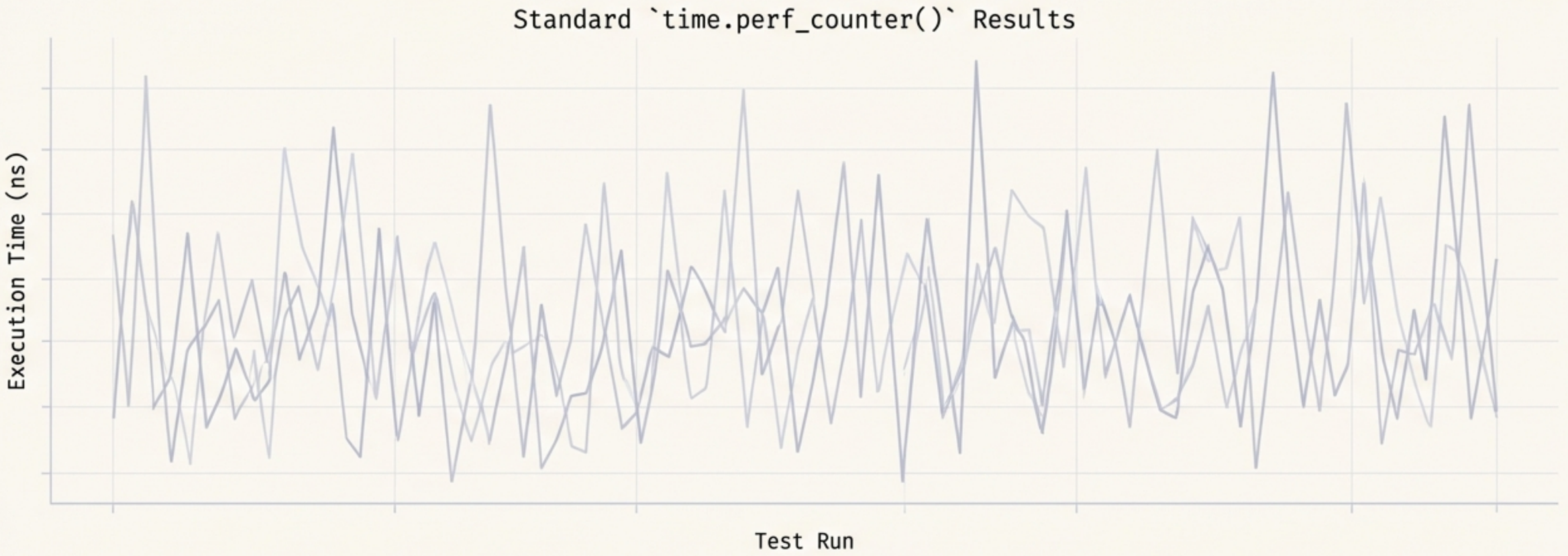
Your Python Benchmarks Are Lying to You



- Standard timing is plagued by noise from the operating system, garbage collection, and context switching.
- The result? Flaky tests, unreliable metrics, and performance changes that are impossible to track.
- You can’t trust the numbers. You can’t build reliable performance tests.

NotebookLM

Achieve Nanosecond-Precision Stability. Every Time.

Introducing `Performance_Measure__Session`: A high-precision benchmarking framework that produces statistically stable, reproducible performance metrics at nanosecond resolution.

It eliminates noise through a rigorous process of Fibonacci-based sampling, outlier removal, and dynamic score normalization.

NotebookLM

From Chaos to Control in Three Lines of Code

1. Import and Measure

```
from osbot_utils.testing.performance
import Perf

def fib(n):
    return n if n < 2 else fib(n-1) +
fib(n-2)

with Perf().measure('fib(10)'):
    fib(10)
```

2. Assert Performance in Tests

```
class Test_MyCode(TestCase):

    def test_fib_10(self):
        with Perf(padding=20) as perf:
            fib(10)

        perf.assert_time(20_000) # asserts
against a normalised score
```

That's it. This single session just ran **1,595 measurements**, removed outliers, and produced a CI-aware, normalised score for a stable assertion.

NotebookLM

Engineered for Trust: Our Design Philosophy

Statistical Robustness: Fibonacci sampling captures warm-up effects; outlier trimming removes GC noise.

Reproducible Scores: Dynamic normalization produces consistent values across runs for stable assertions.

CI-Aware Assertions: Automatic adjustment for slower GitHub Actions runners without code changes.

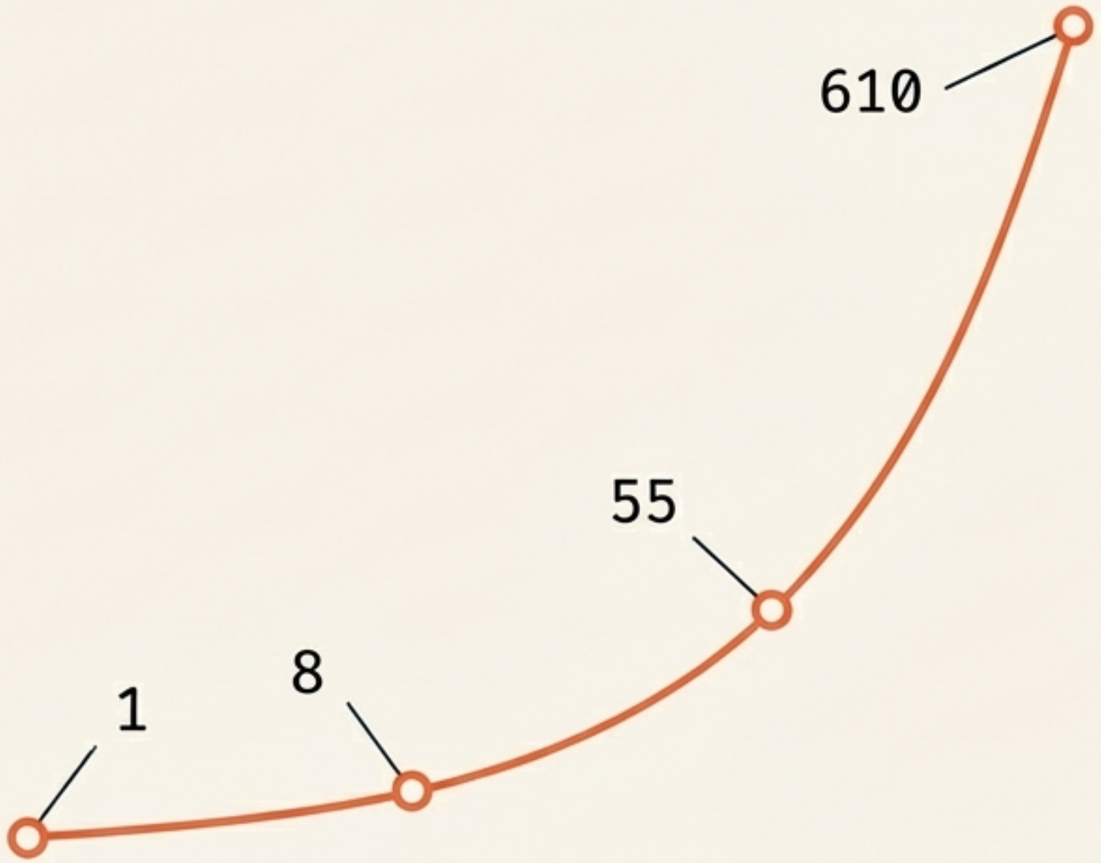
Zero Ceremony: A clean context manager pattern, fluent API, and sensible defaults get you started instantly.

Type_Safe Integration: All internal models use runtime type checking for maximum reliability.

NotebookLM

How We Capture the Full Performance Picture

The Fibonacci Measurement Loop: [1, 1, 2, 3, 5, 8, …, 377, 610]



Rapid Early Sampling: Catches cold-start and JIT behaviour.

Logarithmic Scaling: Captures warm-up and cache effects efficiently.

Large Final Samples: Provides the data needed for statistical stability.

Total Invocations: 1,595
per `measure()` call.

NotebookLM

Raw Power vs. Stable Assertions

`raw_score`

Weighted median-mean after outlier removal (60% median, 40% mean).

Debugging, deep performance analysis, understanding raw machine timing.

Run 1: **1,847 ns** | Run 2: **1,912 ns**

`final_score`

The `raw_score` dynamically normalized to a magnitude-appropriate precision.

Assertions, stable cross-environment comparisons.

Run 1: **2,000 ns** | Run 2: **2,000 ns**

Why two scores? Raw scores reflect slight machine variations. Final scores are stabilised for reliable assertions that don't flap.

NotebookLM

Conquer CI Flakiness. Zero Code Changes.

Performance assertions automatically adjust when a GitHub Actions environment is detected.

How it works:

- `assert_time(*expected): Automatically uses the last expected time × 5 as an upper bound.
- `assert_time__less_than(max_time): Automatically multiplies the threshold by 6.

> Write your tests for your local machine. We handle the rest.

NotebookLM

Core Usage Patterns: From a Script to a Test Suite

Basic Measurement (Context Manager)

For quick, one-off performance checks in a script.

```
with Perf().measure("my_func execution") as perf:
    my_func()
perf.print()
```

Reusable Session (Recommended for Test Suites)

The standard for test classes, preventing re-initialization overhead.

```
class Test_MyClass(TestCase):
    session: Perf

    @classmethod
    def setUpClass(cls):
        cls.session = Perf()

    def test_my_method(self):
        with self.session as _:
            _.measure("my_method")
            self.instance.my_method()
        self.session.assert_time(...)
```

NotebookLM

Advanced Patterns: Measure More Than Just Functions

Pattern 3: Measuring Class Instantiation

Use Case: Pinpointing performance costs in object creation.

```
class MyHeavyClass:
    # ... complex __init__ ...

with Perf().measure(MyHeavyClass):
    pass # The context manager handles the instar
```

Pattern 4: Chained Methods (Fluent API)

Use Case: Creating concise, readable, one-line measurements and assertions.

```
Perf().measure(my_func).print().assert_time(1000)
```

NotebookLM

Taming Slow Functions: Control Your Test Times

For functions running longer than 100ms, a full 1,595-invocation run is overkill. Use a faster mode to balance precision and speed.

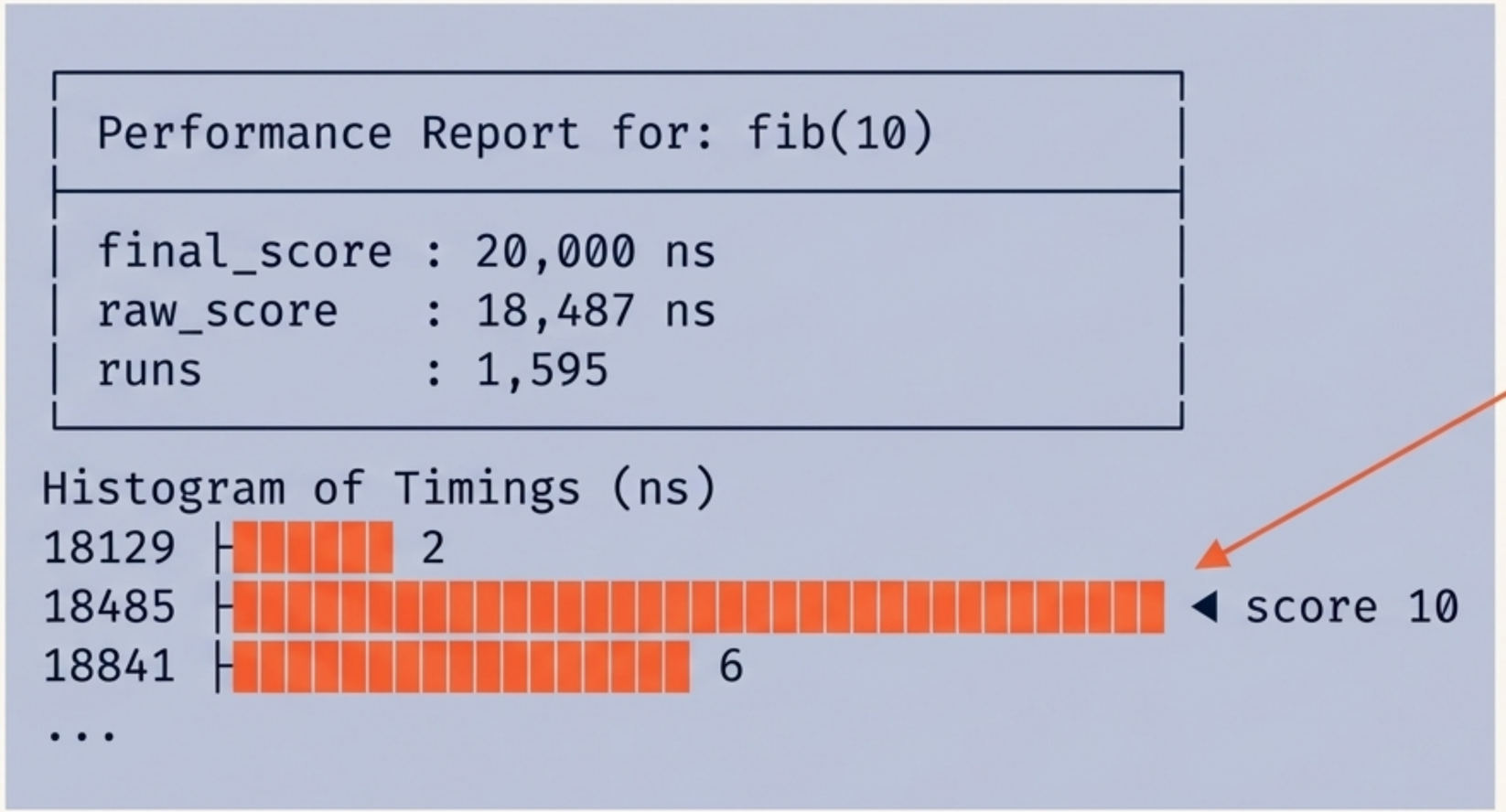
Method	Invocations	Best For
`measure(target)`	1,595	Default, maximum precision for fast functions
`measure__fast(target)`	87	Balanced speed and precision
`measure__quick(target)`	19	Quick checks for very slow functions (>100ms)

```
# Use quick mode for a slow I/O-bound function
with Perf() as perf:
    perf.measure__quick(slow_database_call)
perf.print()
```

NotebookLM

Go Deeper: From a Single Number to the Full Story

Need more than a final score? The detailed report shows the full distribution of your timings.



This marker shows exactly where your calculated score falls within the distribution of all 1,595 measurements.

NotebookLM

Best Practices for Robust Performance Tests

DOs

- **Use a class-level session** in test suites to avoid overhead.
- **Define time thresholds** as named class attributes (e.g., `TIME_FIB_10_NS = 20_000`).
- **Provide multiple expected values** for CI stability: `` `assert_time(20_000, 80_000)` ``.
- **Use** `` `assert_time__less_than` `` for flexible upper-bound checks.

DON'Ts

- **Create new sessions** inside each test method.
- **Measure functions with side effects** (I/O, network calls, randomness).
- **Use raw literal values** in assertions.
- **Skip printing output** when debugging a new test.

NotebookLM

Your Guide to Common Performance Hurdles

Assertion Passes Locally But Fails in CI.

- **Cause:** CI is slower than the automatic 5-6x multiplier allows for your specific test.
- **Solution:** Use `assert_time__less_than(max_value)` for a more generous upper bound, or add a higher expected value value `assert_time(local_val, ci_val)`.

Inconsistent Scores Between Runs.

- **Cause:** The function has variable execution time (I/O, network) or the system is under heavy load.
- **Solution:** Only measure pure, computational functions. Run tests in an isolated environment.

Score is 0ns.

- **Cause:** The function is too fast to measure reliably (sub-100ns).
- **Solution:** Wrap the target function in a loop to measure a larger, more stable unit of work.

NotebookLM

API Quick Reference

Core Session Methods

Method	Description
`measure(target)`	Runs the full 1,595-invocation measurement loop.
`measure__quick(target)`	Quick mode (19 invocations) for slow functions.
`measure__fast(target)`	Fast mode (87 invocations) for balance.
`print()`	Prints the one-line result.
`print_report()`	Prints the detailed report with histogram.
`assert_time(*expected)`	Asserts final score matches an expected value.
`assert_time__less_than(max)`	Asserts final score is below a threshold.

Key Parameters

* `assert_enabled: bool` (Default: `True`): Globally enable/disable all assertion methods.
* `target: Callable`: The function or class to measure.
* `loops: List[int]`: Provide a custom list of loop sizes to override the Fibonacci default.

NotebookLM

Your Checklist for Stable, Reliable Benchmarking

- [ ] Import `Perf` from `osbot_utils.testing.performance`.
- [ ] Create your session once in `setUpClass()`, not per-test.
- [ ] Use the context manager pattern: `with self.session as _:`.
- [ ] Define time thresholds with a clear naming convention.
- [ ] Provide multiple expected values for cross–environment stability (local first, CI last).
- [ ] Use `.print_report()` for deep analyss and `.print()` for quick checks.
- [ ] Use `assert_time()` for exact values and `assert_time__less_than()` for upper bounds.
- [ ] Use `measure__quick()` for slow functions (>100ms).
- [ ] Measure pure functions—no I/O, network, or randomness.
- [ ] Remember: CI gets a 5-6x multiplier automatically.

NotebookLM