Memray Package Report

---

1. Which package/library did you select?

I selected **Memray**, an advanced memory profiler for Python applications.

---

2. What is the package/library?

What purpose does it serve?

Memray helps developers **find where memory is being allocated** in Python programs (including native extension code), so they can:

- diagnose memory leaks
- identify high-allocation hot spots
- compare memory behavior before/after optimizations
- understand memory growth over time

Unlike very lightweight memory snapshots, Memray is designed to produce detailed allocation traces and rich reports (for example, flame graphs and summaries) that make large applications easier to debug.

How do you use it?

There are two common ways:

1. **CLI workflow** (most common)
2. **Python API workflow** (embedded profiling in code)

A. CLI workflow

Run a script under Memray, then generate reports:

```
python -m memray run -o output.bin your_script.py
python -m memray flamegraph output.bin -o flamegraph.html
```

Useful CLI modes include:

- ‘run‘: capture allocations into a ‘.bin‘ profile
- ‘flamegraph‘: create interactive HTML flamegraphs
- ‘table‘: tabular allocation report

- ‘summary‘: high-level overview
- ‘stats‘: numeric statistics
- ‘live‘: terminal UI for live tracking while code runs

B. Python API workflow

Use Memray directly in your code to profile selected blocks:

```
import memray

with memray.Tracker("query_profile.bin"):
    run_expensive_query()
```

This is helpful when you only want to profile one code path (for example, a specific request handler or batch job).

Important concepts

- **Profile capture file (‘.bin‘)** : raw allocation data generated during ‘run‘/‘Tracker‘.
- **Reporter phase** : transform ‘.bin‘ data into human-readable output (HTML or terminal reports).
- **Flamegraph width** wider nodes generally indicate more allocated memory in that path.
- **Native mode** include C/C++ stack frames from extension modules for deeper visibility.

Why Memray is a strong fit for larger Python systems

- It can profile both Python and native allocations.
- It supports multiple reporter styles, not just one view.
- It integrates well with service workflows where you need repeatable profiling artifacts.
- It is practical for ?capture now, analyze later? workflows.

---

3. What are the functionalities of the package/library?

Below are key Memray capabilities, with examples.

Functionality A: Track allocations while code runs

```
python -m memray run -o app_profile.bin app.py
```

Typical output (representative):

```
Writing profile results into app_profile.bin
```

Functionality B: Generate flamegraph reports

```
python -m memray flamegraph app_profile.bin -o app_flamegraph.html
```

Output artifact:

```
app_flamegraph.html
```

Functionality C: Programmatic profiling in application code

```
from memray_tools import profile_function

result = profile_function(run_bulk_queries, profile_prefix="bulk")
print(result["profile"]["profileFile"])
```

Representative output:

```
{
  "profileFile": "bulk_20260408_110201_123456.bin",
  "durationMs": 84.22
}
```

Functionality D: Integrating reports into a web app

In this repository, profiling endpoints generate Memray output and expose flamegraphs in the UI:

- ‘POST /profile/query‘
- ‘POST /profile/bulk-query‘
- flamegraphs served at ‘/flamegraphs/...‘

This allows ?profile + inspect? from the browser without leaving the app workflow.

---

4. When was it created?

Memray was published as an open-source project by Bloomberg on **April 27, 2022**.

---

5. Why did you select this package/library?

I selected Memray because this project is specifically about a **database workload analyzer** where memory behavior matters, and Memray provides practical advantages:

- It creates actionable reports (especially flamegraphs) that map allocations to call paths.
- It supports profiling in realistic workflows (bulk query execution, repeated test runs, browser-driven profiling).
- It is well suited for comparing query-pattern changes (for example, random user generation or large inserts) against memory impact.
- It enables a complete pipeline: **capture -> report -> inspect** .

In short, the choice was driven by **fit for purpose** (debugging memory behavior in Python workloads), not novelty.

---

6. How did learning the package/library influence your learning of the language?

Learning Memray reinforced several Python concepts:

- **Context managers** for deterministic setup/teardown around profiling.
- Better understanding of Python vs native memory behavior in mixed stacks.
- Separation of concerns:
- capture instrumentation in backend code
- report generation as a separate phase
- visualization in frontend/UI
- Performance-aware coding habits (testing assumptions with evidence instead of guessing).

It also improved my ability to reason about resource behavior in addition to pure logic correctness.

---

7. How was your overall experience with the package/library?

Overall experience

Positive overall. Memray is powerful, and once integrated, it gives much clearer visibility into memory-heavy paths than ad-hoc print/debug approaches.

When would I recommend it?

I would recommend Memray when:

- your Python app has unexplained memory growth
- you need to analyze memory under realistic load
- you use native-heavy libraries and want full-stack visibility
- you want report artifacts that can be shared during code reviews or debugging sessions

Would I continue using it?

Yes, especially for backend services and performance-sensitive tools.
I would keep using it because it supports repeatable profiling workflows and produces reports that are easy to inspect and discuss with a team.

References

- Bloomberg announcement (open-source publication date): https://www.bloomberg.com/company/stories/bloomberg-memray-open-source-profiler-python-code/
- Official documentation (getting started): https://bloomberg.github.io/memray/getting_started.html
- Official flamegraph documentation: https://bloomberg.github.io/memray/flamegraph.html
- GitHub repository and command usage overview: https://github.com/bloomberg/memray