

§ 14 PoC 0 結果報告 — Hybrid Bit-Exact Isolate Model

最小実証

項目	値
対象 spec	spec_section_14_draft.md v0.1, § 14.9 PoC 計画 PoC 0
実施日	2026-06-04
環境	WSL Ubuntu 24.04 / Rust 1.93.1 / CPython 3.12.3 / PyO3 0.22.6
目標	「動的 Python を Rust で再発明しない、Python のまま CPython に隔離実行」が本当に bit-exact になるか実機検証
結果	☑PASS(SHA-256 完全一致)、binary size 実測

0. 結論(先に)

Python オリジナルと Rust+CPython-Isolate 同梱バイナリの出力が SHA-256 で完全一致。30 byte 出力、SHA-256 = 8a07ddf0686362c4d446c017ea337e186a8d85b584e93370438f2d2f8b982ae2。

§ 14 Hybrid Bit-Exact Isolate Model の核心主張「動的部分は Python のまま CPython に走らせる、決定論モードで bit-exact 保全」は実機で成立することが確認できました。

1. 何を実証したか

Python の § 13 REJECT 構文(変数名 `getattr` + generator expression による動的属性読み)を含むコードを、以下の 2 系統で実行し、`stdout` の SHA-256 が一致することを確認:

a. 純 Python(リファレンス)

```
class Box: pass
b = Box()
for k, v in [("x", 10), ("y", 20), ("z", 30)]:
    setattr(b, k, v)
total = sum(getattr(b, name) for name in ["x", "y", "z"])
print(f"static=100|dynamic={total}|sum={100 + total}")
```

b. Hybrid バイナリ(本 PoC)

- 静的 Rust 側で `attr_names`、`attr_pairs`、`static_part = 100` を確定
- これらを **canonical encoding**(Python の `list[(str, i64)]`)で Isolate に渡す
- Isolate(埋込 CPython)が 動的 Python ソース断片を実行: `class Box`、`setattr`、`getattr(b, name) for name in ...`

- Isolate から `total: i64` を Rust 側に返す
- Rust 側で `static_part + total` を合成して stdout 出力

2. 実測結果

出力一致

```
py_sha=8a07ddf0686362c4d446c017ea337e186a8d85b584e93370438f2d2f8b982ae2 bytes=30
rs_sha=8a07ddf0686362c4d446c017ea337e186a8d85b584e93370438f2d2f8b982ae2 bytes=30
*** PASS: Python = Rust+CPython-Isolate, SHA-256 bit-exact ***
```

両者の標準出力:

```
static=100|dynamic=60|sum=160
```

バイナリサイズ

```
-rwxrwxrwx 2 root root 348992 Jun  4 16:38 target/release/poc0_hybrid
```

= 348,992 bytes(~341 KB)。これは Rust binary 単体。libpython3.12 は **動的リンク**(システム install 済み):

```
libpython3.12.so.1.0 => /lib/x86_64-linux-gnu/libpython3.12.so.1.0
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6
```

構成	サイズ	配布性
本 PoC(libpython3.12.so を動的リンク)	341 KB Rust + ~5 MB libpython.so(別途)	dev / Linux で Python 入りなら即動く
Hybrid Pro tier 想定(CPython 静的リンク + stdlib bundle)	~30-50 MB single binary	self-contained、商用配布対象
想定 WASM ターゲット(Pyodide subset 同梱)	~15-30 MB(圧縮 ~10-20 MB)+ ~3-5 秒起動	<code>wasmtime</code> 3 行配信

3. 決定論モードの設定

```
std::env::set_var("PYTHONHASHSEED", "0");
```

を `Python::with_gil` の最初の呼び出しより前に設定。pyo3 `auto-initialize` feature は最初の GIL 取得で `Py_Initialize` が走るため、`env` はそれより前に有効化必須。

本 PoC のコードは `random / time / dict iteration` 順序に依存していないため、`PYTHONHASHSEED` なしでも同一出力になりますが、§ 14.2 のポリシーに従って `defensive` に設定しています。

4. 検証コード

a. Rust(`src/main.rs`)抜粋

```
use pyo3::prelude::*;
use pyo3::types::{PyAnyMethods, PyDict, PyDictMethods};
use std::io::Write;

fn main() {
    std::env::set_var("PYTHONHASHSEED", "0");

    let static_part: i64 = 100;
    let attr_names = ["x", "y", "z"];
    let attr_vals: [(&str, i64); 3] = [("x", 10), ("y", 20), ("z", 30)];

    let dynamic_total: i64 = Python::with_gil(|py| -> PyResult<i64> {
        let globals = PyDict::new_bound(py);
        // ... Rust → Python boundary encoding ...
        let code = r#"
class Box:
    pass

b = Box()
for k, v in __rust_attr_pairs:
    setattr(b, k, v)
total = sum(getattr(b, name) for name in __rust_attr_names)
"#;

        py.run_bound(code, Some(&globals), None)?;
        let total: i64 = globals.get_item("total")?.unwrap().extract()?;
        Ok(total)
    }).unwrap();

    let payload = format!("static={}|dynamic={}|sum={}\n",
        static_part, dynamic_total, static_part + dynamic_total);
    std::io::stdout().lock().write_all(payload.as_bytes()).unwrap();
}
```

b. Python リファレンス(`reference/hybrid_demo.py`)

```
def main() -> None:
    static_part: int = 100
    attr_names: list[str] = ["x", "y", "z"]
    attr_pairs: list[tuple[str, int]] = [("x", 10), ("y", 20), ("z", 30)]

    class Box: pass
    b = Box()
    for k, v in attr_pairs:
        setattr(b, k, v)
    total = sum(getattr(b, name) for name in attr_names)
```

```
sum_total = static_part + total
sys.stdout.write(f"static={static_part}|dynamic={total}|sum={sum_total}\n")
```

c. 検証スクリプト(`run_bit_exact.sh`)

```
#!/bin/bash
set -e
cd "$(dirname "$0")"
. /root/.cargo/env
export PYTHONHASHSEED=0
cargo build --release
target/release/poc0_hybrid > /tmp/poc0_rs.out
python3 reference/hybrid_demo.py > /tmp/poc0_py.out
RS_SHA=$(sha256sum /tmp/poc0_rs.out | awk '{print $1}')
PY_SHA=$(sha256sum /tmp/poc0_py.out | awk '{print $1}')
[ "$RS_SHA" = "$PY_SHA" ] && echo PASS || echo FAIL
```

5. 検証された仕様要素

§ 14 要素	検証状況
Static Region は Rust	☑ <code>main()</code> 直書きの Rust が deterministic に動作
Dynamic Isolate に Python ソース断片を保持	☑ Rust の <code>r#"..."#</code> raw string にそのまま埋込、CPython で実行
canonical bit-exact encoding	☑ Rust <code>[(str, i64); 3]</code> → Python <code>list[(str, int)]</code> 経由(本 PoC は手書き、§ 14.3 の canonical 仕様の最小実例)
PYTHONHASHSEED=0 強制	☑ <code>env::set_var</code> を <code>Python::with_gil</code> 前に実行
bit-exact 同一性	☑ Python ≡ Rust+CPython-Isolate の SHA-256 完全一致

未検証(PoC 1～5 で扱う):

- WASI ビルド(`wasm32-wasip1` + Pyodide subset 同梱)
- cross-platform 検証(x86_64 / aarch64 / WASI 三者一致)
- musl-libm pin による libm drift 抑止
- RustPython 版(Light tier)との対比
- 自動 partition(`Step 1.6`)で混在コードを振り分け、本 PoC は手動 partition

6. 重要な観察

1. PyO3 + CPython 同梱はライブラリレベルで非常に軽量: Rust binary 単体は 341 KB と小さく、libpython は dynamic link。Hybrid Pro tier(静的 link で self-contained 配布)が 30-50 MB に膨ら

むのは CPython core + stdlib のサイズ寄与。dev / 評価用なら 341 KB binary + システム Python で済む。

- 2. PyO3 0.22 API のクセ: `py.run_bound(code, globals, locals)` は `&str` を取る(`&CStr` 不可)。raw string `r#"..."#` で複数行 Python ソースを直書き可能。
- 3. 境界エンコーディングは Python の `list[(str, i64)]` で OK: PyO3 が自動で list / tuple / int / str を Python 値へ橋渡し。i64 ↔ Python int は両方向 lossless。
- 4. このサンプルは § 13 REJECT 構文を踏んでいる: 変数名 `getattr` + generator expression は § 13 NON_LITERAL なので Step 0 で REJECT されるはず。Hybrid Isolate ではこれをそのまま受理して実行できることが PoC で実証された。

7. 次のステップ(§ 14.9 PoC ロードマップ)

Phase	内容	期間目安
☑PoC 0	最小ビルド + bit-exact 検証(本書)	完了
PoC 1	Any を含む 5 sample を Isolate 化、Python 純実行と bit-exact 比較	1-2 日
PoC 2	*args/**kwargs + 動的 getattr + 動的 monkey patching の各 5 sample	2-3 日
PoC 3	partition(関数単位)の自動振り分け実装、混在サンプル 10 件	3-5 日
PoC 4	musl-libm pin、cross-platform 検証(x86_64 / aarch64 / WASI)	2-3 日
PoC 5	RustPython 版(Light tier)で同一サンプル実行、差分計測	2-3 日
§ 14 v0.2 spec	PoC 0-5 実測値を反映、保証文言を実数値で固定	1 日

8. 配布物

```
engine/poc0_hybrid/  
├─ Cargo.toml           pyo3 0.22 + auto-initialize、release profile pin  
├─ src/main.rs          Static Rust + PyO3 Dynamic Isolate のデモ  
├─ reference/hybrid_demo.py 同等の純 Python リファレンス  
├─ run_bit_exact.sh      ビルド → 実行 → SHA-256 比較  
└─ target/release/poc0_hybrid 348,992 byte の Hybrid バイナリ
```

PoC 0 完了。 § 14 Hybrid Bit-Exact Isolate Model の核心(動的 Python を CPython に隔離実行して bit-exact 維持)が実機で成立することを確認。

PoC 1 以降(Any / *args / 動的 monkey patching の各サンプルでの bit-exact)へ続行可能。