Source: authored in b3u_use_cases as this use case's guide.

Benchmarking the Rust consumer

The MeTTa benchmark document reports `WK_rust`, `MAI_rust` and `WK+MAI_rust` — a walk, an insert, and the two combined, measured through a Rust consumer. This guide runs those same benchmarks on your own data.

You will not write or download any benchmark code. It ships inside the SCB-API Rust package.

1. What you need

Three things, all of which b3u.dev already gives you:

- **The Rust SCB API** — `scb-api-rust.zip` from the packages page. It contains the crate and its examples, including `atom_bench` and `walk_bench`. The crate has no external dependencies.
- **Your grammar's Rust keyword definition** — `mettaKeyWordDefinition.rs`, in the `rust/` directory of your metta bundle.
- **A compiled instance** — any `.bfgr` your frontend produced.

Plus a Rust toolchain (`cargo`).

2. Produce an instance

If you have not already, from the `metta` use case's layout:

```
cd ~/ccsUseCases/metta
./ccsmetta -lt sample.metta
```

⚠ The `-lt` is required and its absence is silent — see the `metta` build guide §5.1. A `.bfgr` of about 224 bytes means the frontend parsed almost nothing and exited 0 anyway.

3. Run the benchmarks

```
unzip -q ~/Downloads/scb-api-rust.zip -d ~/ccsUseCases/scb-api-rust
cd ~/ccsUseCases/scb-api-rust

export CCS_BENCH_KWD=~/ccsUseCases/metta/rust/mettaKeyWordDefinition.rs

cargo run --release --example atom_bench -- 100 \
    ~/ccsUseCases/metta/sample.metta.bfgr
```

Arguments are order-independent: a bare integer is the iteration count, anything else is a `.bfgr`. Pass several to compare them.

`walk_bench` takes the same arguments and reports the walk alone.

4. What comes out

```
corpus,regime,atoms,iters,median_us,mean_us,min_us,max_us
sample.metta.bfgr,WK_rust,-,100,1.86,2.06,1.78,4.94
sample.metta.bfgr,MAI_rust,5,100,0.95,1.14,0.84,3.30
sample.metta.bfgr,WK+MAI_rust,5,100,2.54,2.70,2.38,4.99
```

CSV on stdout, progress on stderr — so `> out.csv` gives you a clean file.

- **WK_rust** — walking the loaded binary, no atoms built.
- **MAI_rust** — inserting atoms that were built outside the timed region.
- **WK+MAI_rust** — both in one pass: the realistic consumer cost, and the column the benchmark document's ratios use.

The binary is loaded once, before timing starts. That split is the whole point of the format.

⚠ 5. Two counts that look contradictory and are not

The `metta_bench` use case's C++ walker reports **23 atoms** for this sample. `atom_bench` reports **5**.

Both are right. They count different things: the Rust bench counts top-level forms — this sample has five s-expressions — while the C++ walker counts leaf tokens, every variable, symbol, string and integer inside them. Neither is *the* atom count; "atom" is a word each consumer defines for itself.

It is worth knowing because two published use cases reporting different numbers for one file looks like a bug, and it is not.

⚠ 6. What these numbers are not

They are not the published figures. The benchmark document's table comes from one machine, one run, and three corpora we chose. Your hardware, your data and your build differ.

More usefully: a small sample tells you almost nothing. The published crossover analysis — produce once, read many times — only becomes visible on inputs large enough for the per-read cost to matter. Run these against your real corpus, not against `sample.metta`.

Section 7 of *MeTTa bulk load — a benchmark* lists what the published result does and does not establish. Read it before quoting any number, including your own.

Terms