

User Manual

1. Overview

`tsdistances` is a Python library with a Rust backend for efficient time-series distance computations. This manual describes how to install and use the code directly from the ACM TOMS ZIP archive, and how to reproduce the results in the accompanying manuscript.

2. Package Layout (Top Level)

The ZIP archive contains only:

- `userManual.pdf`
- `userManual/` (sources used to build `userManual.pdf`)
- `Python/` (Python bindings, tests, datasets, and drivers)
- `Rust/` (Rust implementation)
- `Matlab/` (optional MATLAB bindings)

To rebuild `userManual.pdf`, run `userManual/build_userManual.sh` (requires `pandoc` and a LaTeX engine).

3. Dependencies and Supported Platforms

3.1 Supported Platforms

- Linux (tested)
- macOS (tested)

3.2 Required Tools

- Python 3.12
- Rust nightly toolchain `nightly-2025-06-30` (see `Rust/rust-toolchain.toml`)
- A C/C++ build toolchain suitable for your OS

3.2.1 Linux System Packages

On Debian/Ubuntu, the following packages are required to build from source:

- `build-essential`
- `clang`
- `cmake`
- `curl`
- `git`
- `libssl-dev`
- `libx11-dev`
- `libxcb1-dev`
- `ninja-build`
- `pkg-config`

Example install command (Debian/Ubuntu):

```
sudo apt-get update
sudo apt-get install -y build-essential clang cmake curl git libssl-dev libx11-dev libxcb1-c
```

3.2.2 macOS System Packages

On macOS, install the developer tools and build dependencies:

```
xcode-select --install
brew install cmake ninja llvm pkg-config python@3.12
```

Optional GPU support on macOS requires MoltenVK (via the LunarG Vulkan SDK). Homebrew does not currently provide a `vulkan-sdk` cask. Download and install the macOS SDK from LunarG, then set up the environment:

```
source "$HOME/VulkanSDK/<version>/setup-env.sh"
vulkaninfo --summary
```

If `vulkaninfo` fails or reports an unsupported Vulkan version, GPU tests are not available on that machine.

3.2.3 Rust Components

The default build uses the `tsdistances_gpu` dependency (for GPU kernels) and requires additional Rust components:

```
rustup component add rust-src rustc-dev llvm-tools
```

3.3 Python Dependencies

Core runtime dependencies (from `Python/pyproject.toml`):

- `numpy>=1.21.0,<2.3.0`
- `typeguard`

Testing and reproducibility dependencies:

- `pytest`
- `aeon`
- `stumpy`
- `pandas`

Plotting (for figure regeneration):

- `matplotlib`
- `seaborn`

3.4 Optional GPU Support

For GPU builds and tests, a Vulkan-capable GPU and Vulkan drivers are required. The recommended setup is:

- LunarG Vulkan SDK
- SPIR-V tools (either installed system-wide or using compiled tools via the `use-compiled-tools` feature)

4. Installation and Setup

4.1 From the ZIP Archive (Recommended)

These steps install and run the code directly from the ZIP archive without downloading external code or data.

1. Unzip the archive and enter the Python package directory.

```
cd tsdistances/Python
```

2. Create and activate a virtual environment.

```
python3 -m venv .venv
source .venv/bin/activate
```

3. Install build tooling.

```
pip install --upgrade pip
pip install maturin
```

4. Build and install the package from the local sources. The Rust crate is in `../Rust` and is referenced by `pyproject.toml`.

```
maturin develop --release
```

5. Run the deterministic driver program (no interactive input required).

```
python drivers/driver_quickstart.py
```

The output should match `drivers/driver_quickstart.expected.txt`.

6. Run the correctness tests.

```
pip install pytest aeon stumpy
pytest -v tests/test_correctness_cpu.py
```

4.2 From Source (CPU and GPU Builds)

- CPU build is identical to the steps in Section 4.1.
- GPU build requires Vulkan drivers and SPIR-V tools. Use the compiled tools feature for convenience.

```
maturin develop --release --features use-compiled-tools
```

4.3 PIP (Online, Optional)

This method is not suitable for offline reproducibility, but is provided for convenience.

```
pip install tsdistances
```

5. Usage

5.1 Example: DTW on CPU and GPU

```
import numpy as np
import tsdistances

x1 = np.random.rand(100)
x2 = np.random.rand(100)

cpu_distance = tsdistances.dtw_distance(x1, x2, device='cpu')
print(cpu_distance)

# Requires Vulkan build
# gpu_distance = tsdistances.dtw_distance(x1, x2, device='gpu')
# print(gpu_distance)
```

5.2 Example with Expected Output

```
python drivers/driver_quickstart.py
```

Expected output:

```
erp_distance: 2
dtw_distance: 1
lcss_distance: 0.3333333333333333
```

6. User Callable Procedures (API Summary)

All distance functions accept 1-D arrays (single series) or 2-D arrays (sets of series). If *v* is omitted, pairwise distances within *u* are computed. For GPU-enabled functions, *device* must be "cpu" or "gpu".

Common parameters:

- *u*: array-like, shape (N,) or (M, N)
- *v*: array-like, optional
- *band*: Sakoe-Chiba band in [0, 1] where applicable
- *par*: True to use all CPU cores, False for single thread
- *device*: "cpu" or "gpu" (GPU requires a Vulkan build)

Functions:

- *euclidean_distance(u, v, par=True)*: Euclidean distance.
- *catcheucl_distance(u, v, par=True)*: CATCH22 Euclidean distance.
- *erp_distance(u, v=None, band=1.0, gap_penalty=0.0, par=True, device="cpu")*: ERP distance.

- `lcss_distance(u, v=None, band=1.0, epsilon=1.0, par=True, device="cpu")`: LCSS distance.
- `dtw_distance(u, v=None, band=1.0, par=True, device="cpu")`: DTW distance.
- `ddtw_distance(u, v=None, band=1.0, par=True, device="cpu")`: Derivative DTW.
- `wdtw_distance(u, v=None, band=1.0, g=0.05, par=True, device="cpu")`: Weighted DTW.
- `wddtw_distance(u, v=None, band=1.0, g=0.05, par=True, device="cpu")`: Weighted Derivative DTW.
- `adtw_distance(u, v=None, band=1.0, warp_penalty=1.0, par=True, device="cpu")`: Amerced DTW.
- `msm_distance(u, v=None, band=1.0, par=True, device="cpu")`: Move-Split-Merge distance.
- `twe_distance(u, v=None, band=1.0, stiffness=0.1, penalty=0.1, par=True, device="cpu")`: Time Warp Edit distance.
- `sb_distance(u, v=None, par=True)`: Shape-Based distance.
- `mp_distance(u, window, v=None, par=True)`: MPDist (requires window).

6.1 Parameter Guidance

- **band**: Use smaller values for speed and to constrain warping; **band=1.0** is unconstrained.
- **g** (WDTW/WDDTW): Controls the stiffness of the weighting; larger values penalize warping more.
- **warp_penalty** (ADTW): Non-negative; increases cost of warping.
- **stiffness** and **penalty** (TWE): Control elastic and deletion costs.

7. Reproducibility

7.1 Datasets Included

All datasets required for the benchmark table are included under `Python/datasets/ucr`. No additional downloads are needed.

7.2 Correctness Tests

CPU correctness against AEON:

```
pip install pytest aeon stumpy
pytest -v tests/test_correctness_cpu.py
```

GPU correctness (requires Vulkan build):

```
pytest -v tests/test_correctness_gpu.py
```

7.3 Benchmark Table Reproduction

The benchmark table reported in the paper can be reproduced with the included script. This runs DTW on the UCR datasets and reports single-thread, parallel, and optional GPU timings.

```
python scripts/benchmark_ucr.py --mode dtw --runs 1 --markdown
```

Results are written to `benchmarks/benchmark_summary.csv` by default. Timing values vary by hardware.

For comparison with the paper table, run:

```
python scripts/benchmark_ucr.py --mode dtw --runs 1 --paper-format --gpu
```

The expected paper table format is provided in `scripts/benchmark_ucr.expected.csv` (values depend on hardware).

To run the same benchmark logic as `tests/test_benchmark.py` on UCR datasets (ERP/DTW/ADTW vs AEON), use:

```
python scripts/benchmark_ucr.py --mode full --runs 1 --gpu
```

7.4 Plotting the Dataset Figure

To regenerate the dataset scatter plot, install plotting dependencies and run the script below.

```
pip install matplotlib seaborn pandas
python tests/plot_journal.py
```

8. Drivers and Expected Output

Driver programs are provided under `Python/drivers/`. Each driver is accompanied by a file with expected output:

- `drivers/driver_quickstart.py`
- `drivers/driver_quickstart.expected.txt`

These drivers do not require interactive input.

9. Directory Layout (Detailed)

- `Python/`
 - `pyproject.toml` (Python build metadata; references Rust crate)
 - `python/` (Python package sources)
 - `tests/` (unit and correctness tests)
 - `datasets/` (UCR datasets used in the benchmarks)
 - `scripts/` (benchmark and reproduction scripts)
 - `drivers/` (driver programs and expected outputs)
- `Rust/`
 - `Cargo.toml`, `Cargo.lock`, `src/` (Rust implementation)

- `Matlab/` (optional MATLAB bindings)
- `userManual/` (sources for this manual)

10. Troubleshooting

- If Rust build fails, install the required toolchain explicitly:

```
rustup toolchain install nightly-2025-06-30
```

- If GPU tests fail, verify Vulkan drivers are installed and rebuild with GPU features (Section 4.2).
- If you see a warning about failing to set `rpath` during `maturin` builds, install `patchelf` and retry:

```
sudo apt-get install -y patchelf
```

```
# or:
```

```
pip install patchelf
```