Modular Python Repository Architecture for Pressure–Oscilloscope Dataset Processing, Alignment, Adapters, and Visualization

1 Goals & Requirements (from the Specification)

- Two unsynchronized streams: P-stream (timestamps + voltages $\rightarrow$ scalar pressure $p^{(3)}$) and O-stream (per-file time series with N_F, sampling step δt_F).
- File-level midpoint alignment to nearest P-stream timestamp; acceptance threshold $O_{\max}$; uncertainty bound via local derivative $\widehat{\dot{p}}_W$.
- In-memory tables for `Signals`, `OscFiles`, `File2PressureMap`; **fits in memory**, but offer optional lazy/batched loading for large corpora.
- Framework-agnostic outputs (NumPy-first), ready for PyTorch/TF dataset wrappers at a later stage.
- **Hydra**-driven configuration via YAML (hierarchical, overrideable).
- **Extremely modular adapters**: each in its own folder with `adapter.py`, `adapter.yml`, `README.md` (math + references), and `tests/`.
- Visualization: quick CLI flags to plot raw/processed/adapter outputs (e.g., random samples at given pressure ranges) or return NumPy arrays.
- Trackability: every record carries keys `sid`, `file_stamp`, `idx` (and run UUID); deterministic tie-breaking and seeding.

2 High-Level Architecture

- **Ingestion & Indexing**: resolve dataset paths, parse file names and per-record stamps, build in-memory registries.
- **Calibration & Mapping**: compute $p^{(3)}$ from voltages, perform midpoint alignment to nearest P-stream time, compute E_{align} and $|\Delta P|$ bounds.
- **Adapters Layer 1**: cycle-synchronous and fixed-length, shift-invariant mappings (PB-CSA, PLSTN, HMV, CEC, DTW-TA, MTP).
- **Adapters Layer 2 (Transforms)**: FT spectrum, Hilbert-envelope, wavelet energies, MFCC.
- **Visualization**: reusable plotting and inspection utilities.
- **Export (on-demand)**: produce *in-memory* datasets $\{(\boldsymbol{x}_F, p^*(F))\}$ for ML; optional save-as you debug (CSV/NPZ).

3 Repository Layout (copy/paste)

```
repo-root/
 pyproject.toml                 # PEP 621 metadata; dependencies & tooling
 README.md                      # project overview; quickstart
 LICENSE
 .gitignore
 .pre-commit-config.yaml        # black/ruff/isort/mypy hooks
 conf/                          # Hydra configs (hierarchical)
   config.yaml                 # master config: paths, policies, defaults
   dataset/                    # dataset-related config group
     default.yaml             # root paths; timezone; timestamp grammar
     dev.yaml                 # overrides for small/dev runs
   mapping/                    # alignment policies and thresholds
     default.yaml             # O_max, tie_breaker, W, kappa...
   calibration/
     default.yaml             # alpha,beta; units; scalar_channel=3
   adapter/                    # choose adapter & hyperparams
     pb_csa.yaml
     plstn.yaml
     hmv.yaml
     cec.yaml
     dtw_ta.yaml
     mtp.yaml
     fts.yaml
     hte.yaml
     wcv.yaml
     mfcc.yaml
   viz/
      default.yaml             # plotting defaults; styles; limits
 data/                          # (user-provided path via config)
   raw/                        # untouched raw inputs
   scratch/                    # OPTIONAL debug outputs only
 docs/
   architecture.md
   adapters/
     PB-CSA.md
     PLSTN.md
     HMV.md
     CEC.md
     DTW-TA.md
     MTP.md
     FTS.md
     HTE.md
     WCV.md
     MFCC.md
   references.bib              # if you build Sphinx/MkDocs w/ bib support
 src/
   echopress/                  # project package (importable)
      __init__.py
      cli.py                   # Typer/Click entrypoints + Hydra main
```

```
    types.py                 # dataclasses / TypedDict Protocols
    utils/                   # small helpers (time, signal, io)
      __init__.py
      timeparse.py          # parse M..-D..-H..-M..-S..-U.xxx
      signals.py            # basic DSP utilities
      windows.py            # tapers, resampling helpers
      logging.py            # structlog or std logging config
    ingest/
      __init__.py
      indexer.py            # walk folders; build registries
      pstream.py            # parse P-stream files to [(T^P, p)]
      ostream.py            # parse O-stream files to arrays + meta
    core/
      __init__.py
      calibration.py        # p^(k)=alpha_k v^(k)+beta_k; scalar p=channel 3
      mapping.py            # midpoint alignment; E_align; tie-break
      derivative.py         # DerivEst (central/local-linear/savgol)
      uncertainty.py        # |P| <= kappa |dp/dt| * E_align
      tables.py             # build in-memory tables (Signals, Map, ...)
    adapters/                # unified adapter interface + library
      __init__.py
      base.py               # Adapter Protocol; registry; validation
      pb_csa/
        adapter.py
        adapter.yml
        README.md
        tests/
      plstn/ ...            # same pattern for each adapter
      hmv/
      cec/
      dtw_ta/
      mtp/
      fts/
      hte/
      wcv/
      mfcc/
    viz/
      __init__.py
      plot_signals.py       # raw vs calibrated vs pressure overlay
      plot_alignment.py     # show T_mid vs nearest T^P; errors
      plot_adapter.py       # grid of adapter outputs, random by pressure
      styles.py
    export/
       __init__.py
       to_numpy.py          # (X, y) in-memory build; optional save
       datasets.py          # Torch/TF-ready wrappers (future)
tests/                       # pytest suite
  test_ingest.py
  test_mapping.py
  test_derivative.py
  test_uncertainty.py
```

```
   test_tables.py
   adapters/
      test_pb_csa.py
      test_plstn.py
      ...
      test_mfcc.py
```

4 Configuration with Hydra (YAML)

Master config `conf/config.yaml`

```yaml
defaults:
  - dataset: default
  - calibration: default
  - mapping: default
  - adapter: fts         # choose active adapter group by default
  - viz: default
  - _self_

run:
  seed: 1234
  num_workers: 0

paths:
  data_root: ${dataset.data_root}
  pstream_glob: "${dataset.data_root}/raw/pstream/*.txt"
  ostream_glob: "${dataset.data_root}/raw/ostream/*.csv"
  tz: "UTC"
```

Example `conf/mapping/default.yaml`

```yaml
O_max: 0.250            # seconds
tie_breaker: "earliest" # or "latest"
W: 2.0                  # seconds, derivative window
kappa: 1.0
derivative:
  method: "central_diff"    # local_linear | savgol
  min_records_in_W: 3
reject_if_Ealign_gt_Omax: true
```

Example adapter config `conf/adapter/fts.yaml`

```yaml
name: "fts"
output_length: null         # default = M/2+1 for real-valued segment
segment:
  length: 2048              # Ms
  hop: null                 # optional, if you want sliding segments
detrend: "linear"           # or "none"
window: "hann"
normalize: "l2"             # none | l2 | dB_ref
```

5 Typed Data Model (in-memory tables)

Define simple, typed containers (Python `dataclasses`) to hold the information computed per file/record; keys ensure traceability:

- `OscFile`: `sid`, `file`$_s tamp(\mathtt{T}_F^{\mathtt{start}})$, N_F, δt_F, D_F, $T_F^{\mathtt{mid}}$.
- `Signals`: rows keyed by (`sid`, `file_stamp`, `idx`) with $t_{F,n}$ and channel amplitudes.
- `File2PressureMap`: (`sid`, `file_stamp`, $T^{P*}, p^*, E_{\mathtt{align}}, |\Delta P|, O_{\max}, W, \kappa$).
- `AdapterVector`: (`sid`, `file_stamp`, `adapter_name`, $\boldsymbol{x}_F \in \mathbb{R}^L$).

Skeleton (copy/paste)

```
# src/echopress/types.py
from dataclasses import dataclass
from typing import Tuple, Optional, Dict, Any
import numpy as np

Key = Tuple[str, str]  # (sid, file_stamp)

@dataclass(frozen=True)
class OscFile:
    sid: str
    file_stamp: str          # ISO8601-like or raw "M..-D..-H..-M..-S..-U.xxx"
    N: int
    delta_t: float
    duration: float
    t_mid: float             # absolute seconds since epoch

@dataclass
class File2PressureMap:
    sid: str
    file_stamp: str
    p_time: float            # T^{P*}
    p_value: float           # p^*(F)
    E_align: float
    dP_bound: float
    O_max: float
    W: float
    kappa: float

@dataclass
class SignalsRow:
    sid: str
    file_stamp: str
    idx: int
    t_abs: float
    v: np.ndarray            # shape (C,), channel amplitudes

@dataclass
class AdapterVector:
```

```
    sid: str
    file_stamp: str
    adapter: str
    x: np.ndarray            # shape (L,)
```

6 Core Pipeline Modules (responsibilities & signatures)

Ingest

```
# src/echopress/ingest/indexer.py
def index_dataset(p_glob: str, o_glob: str) -> Dict[str, Any]:
    """Walk paths with pathlib, return dict of discovered P-stream/O-stream files,
    grouped by sid, with parsed timestamps and simple stats."""
```

Pressure parsing, calibration

```
# src/echopress/ingest/pstream.py
def load_pstream(path: str, calib: Dict[str, Any]) -> np.ndarray:
    """Return sorted array [(T^P_m, p_m)], with p_m = alpha_3*v^(3)+beta_3."""
```

Oscilloscope loading

```
# src/echopress/ingest/ostream.py
def load_ostream_file(path: str) -> Tuple[np.ndarray, Dict[str, Any]]:
    """Return (t_abs: (N,), v: (N,C)), plus meta (sid, start stamp, delta_t ...)."""
```

Mapping, derivative, uncertainty

```
# src/echopress/core/mapping.py
def nearest_pressure(t_mid: float, P: np.ndarray, tie_breaker: str) -> Tuple[float,float]:
    """Return (T^{P*}, p(T^{P*})) by nearest timestamp; break ties deterministically."""

def align_error(t_start: float, N: int, delta_t: float, T_Pstar: float) -> float:
    """E_align = | t_start + 0.5*N*delta_t - T^{P*} |"""

# src/echopress/core/derivative.py
def deriv_est(P: np.ndarray, T_Pstar: float, W: float, method: str, min_pts: int) -> float:
    """Estimate dp/dt at T^{P*} with a window W, using method (central_diff, local_linear, sa

# src/echopress/core/uncertainty.py
def pressure_bound(dpdt: float, E_align: float, kappa: float) -> float:
    """|P| <= kappa * |dp/dt| * E_align"""
```

Tables assembly

```
# src/echopress/core/tables.py
def build_tables(...)-> Dict[str, Any]:
    """Construct in-memory 'Signals', 'OscFiles', 'File2PressureMap' from ingest+core."""
```

7 Adapter Framework

Interface & registry

```
# src/echopress/adapters/base.py
from typing import Protocol, Mapping
import numpy as np

class Adapter(Protocol):
    def name(self) -> str: ...
    def fit(self, v: np.ndarray, t: np.ndarray, cfg: Mapping) -> "Adapter": ...
    def transform(self, v: np.ndarray, t: np.ndarray, cfg: Mapping) -> np.ndarray: ...
    def fit_transform(self, v: np.ndarray, t: np.ndarray, cfg: Mapping) -> np.ndarray: ...

ADAPTERS = {}  # name -> constructor

def register(name: str):
    def deco(cls):
        ADAPTERS[name] = cls
        return cls
    return deco
```

Per-adapter folder contract

Every adapter folder contains:

- `adapter.py` (implements `Adapter` protocol, registers itself)
- `adapter.yml` (hyperparameters)
- `README.md` (theory, math, references)
- `tests/` (unit tests on synthetic signals)

8 Visualization Module

```
# src/echopress/viz/plot_adapter.py
def grid_by_pressure(adapter_name: str, pr_min: float, pr_max: float, n: int, seed: int, ...)
    """Sample 'n' files with p*(F) in [pr_min, pr_max], compute adapter vectors,
       and plot (or return) results. Supports --return_numpy to emit arrays."""
```

9 Export (on-demand, for ML)

```
# src/echopress/export/to_numpy.py
def build_xy(adapter_name: str, subset_keys=None, save: bool=False, path: str=None):
    """Return X: (B, L), y: (B,), where B = number of files selected;
       If save=True, store NPZ for debugging; otherwise in-memory only."""
```

10 CLI Entrypoints (Hydra + Typer)

```
# src/echopress/cli.py
import typer
import hydra
```

```
from omegaconf import DictConfig
app = typer.Typer()

@hydra.main(config_path="../../conf", config_name="config", version_base=None)
def main(cfg: DictConfig):
    app()

@app.command()
def index():
    """Index dataset and print summary."""

@app.command()
def align():
    """Build tables with alignment, derivative, uncertainty."""

@app.command()
def adapt(adapter: str = typer.Option(...),
          pr_min: float = 0.0, pr_max: float = 300.0, n: int = 8,
          return_numpy: bool = False):
    """Plot or return adapter outputs for random files within pressure range."""
```

Example runs

```
# index and alignment with defaults
python -m echopress.cli index
python -m echopress.cli align mapping.O_max=0.200 mapping.tie_breaker=latest

# visualize 9 MFCC vectors for pressures in [80, 120] mmHg
python -m echopress.cli adapt --adapter mfcc --pr-min 80 --pr-max 120 --n 9

# build an in-memory dataset of (FTS, pressure)
python -m echopress.cli adapt --adapter fts --return-numpy true
```

11 Loading Modes (memory vs. lazy)

Default mode eagerly builds all tables in memory (fits dataset *today*). When data grows:

- **Batch iterators**: wrap O-stream file iteration into generators yielding batches of files; tables are materialized *per-batch* for debugging and released after.
- **Selective subsetting**: CLI flags to filter by `sid`, date ranges, or pressure bands before adapter computation.

No persistent database is required; optional scratch artifacts (CSV/NPZ) can be emitted during debugging and then discarded.

12 Testing Strategy

- **Unit tests**: ingest (parsing stamps), mapping (nearest time), derivative estimators (finite-difference truth), uncertainty bound monotonicity.
- **Adapter tests**: synthetic signals with known spectra or periodicity to validate shape, invariance to circular shifts, and output dimension L.

- **Property-based tests** (optional): Hypothesis for shift-invariance (random circular shifts $\Rightarrow$ same features up to numerical tolerance).
- **Determinism**: fixed random seeds; stable tie-breakers.

13 Minimal Per-Module `README.md` Snippets (copy/paste)

Dataset Ingest (`src/echopress/ingest/README.md`)

```
# Ingest Module

Parses P-stream and O-stream files, builds in-memory registries.

- P-stream: timestamp lines "M..-D..-H..-M..-S..-U.xxx" + voltage triple (V).
  - Calibrate p^(3) = alpha_3 v^(3) + beta_3.
- O-stream: per-file uniform sampling with N_F and delta_t_F.

Outputs:
- OscFile table
- Signals table (keyed by (sid, file_stamp, idx))
- Ready for midpoint alignment and uncertainty bound.

References: Oppenheim & Schafer (1989).
```

Mapping & Uncertainty (`src/echopress/core/README.md`)

```
# Mapping & Uncertainty

For each O-stream file F:
- Midpoint T_mid = T_start + 0.5 N_F delta_t_F.
- Nearest P-stream timestamp T^{P*} by absolute difference; deterministic tie-break.
- Alignment error E_align = |T_mid - T^{P*}|.
- Derivative estimate dp/dt at T^{P*} using window W (central, local-linear, or Savitzky-Gola
- Pressure bound: |P| <= kappa * |dp/dt| * E_align.

Config: conf/mapping/default.yaml

References: Oppenheim & Schafer (1989).
```

Adapter Example: HMV (`src/echopress/adapters/hmv/README.md`)

```
# Harmonic Magnitude Vector (HMV)

Idea: Magnitudes at first H harmonics of estimated f0 yield a compact, shift-invariant
representation (phase removed).

Math:
X(k) = | _n v[n] exp(-i 2 k f0 t[n]) |, k=1..H
Feature vector x = [X(1), ..., X(H)]  R^H.

Config: adapter.yml (H, normalization, f0 estimation method).
```

```
Pros: robust to misalignment; compact.
Cons: loses phase/shape.

References: Oppenheim & Schafer (1989).
```

Transform Example: FTS (src/echopress/adapters/fts/README.md)

```
# Fourier Transform Spectrum (FTS)

Idea: Use |DFT| magnitudes (first M/2+1 bins) for a length-M segment of v.

Shift invariance: circular shift affects only phase → magnitudes unchanged.

Config: segment.length (Ms), window, detrend, normalization.

References: Oppenheim & Schafer (1989).
```

Visualization (src/echopress/viz/README.md)

```
# Visualization

- plot_signals.py: raw channels vs. calibrated pressure overlay by absolute time.
- plot_alignment.py: show T_mid vs nearest T^P (scatter), E_align histogram.
- plot_adapter.py: grid of adapter outputs sampled by pressure range; return NumPy.

CLI examples:
python -m echopress.cli adapt --adapter mfcc --pr-min 80 --pr-max 120 --n 9
```

14 Best Practices (tooling & style)

- **Packaging**: `pyproject.toml` with PEP 621; `src/` layout; import as `echopress`.
- **Lint/Format/Type**: ruff + black + isort + mypy via pre-commit.
- **Docs**: Sphinx or MkDocs in `docs/`; math and references for each adapter.
- **Reproducibility**: Hydra config composition; store resolved config in run dir; fixed `seed`.
- **Agnostic ML**: expose `numpy` arrays; thin wrappers for Torch/TF added later.

15 State-of-the-Art, Scientifically Grounded Adapters (recap)

- Cycle-synchronous: PB-CSA [1], PLSTN [2], HMV (Fourier series truncation) [3], CEC/cepstrum [4], DTW-TA [5], MTP/matched filter [6].
- Transforms (time-domain inputs): FT spectrum [3], Hilbert-envelope [7], wavelets [8], MFCC [9].

16 Minimal End-to-End Flow (step list)

1. **Point** `conf/dataset/default.yaml` to dataset root; set timestamp grammar/timezone.
2. **Index**: `python -m echopress.cli index`.

3. **Align**: `python -m echopress.cli align` (build tables; compute E_{align}, $|\Delta P|$).
4. **Inspect**: `python -m echopress.cli adapt -adapter fts -pr-min 80 -pr-max 120 -n 12`.
5. **Export (on-demand)**: `python -m echopress.cli adapt -adapter mfcc -return-numpy true`.
6. **Debug**: toggle Hydra overrides for thresholds/hyperparameters; run unit tests with `pytest -q`.

References

References

[1] S. Braun, *The Extraction of Periodic Waveforms by Time Domain Averaging*, Acustica **32**(2), 69–77 (1975).

[2] P. D. McFadden, *Interpolation techniques for time domain averaging of gear vibration*, Mechanical Systems and Signal Processing **3**(1), 87–97 (1989).

[3] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice Hall (1989).

[4] A. M. Noll, *Cepstrum pitch determination*, J. Acoust. Soc. Am. **41**(2), 293–309 (1967).

[5] H. Sakoe and S. Chiba, *Dynamic programming algorithm optimization for spoken word recognition*, IEEE Trans. Acoust. Speech Signal Process. **26**(1), 43–49 (1978).

[6] G. L. Turin, *An introduction to matched filters*, IRE Trans. Information Theory **6**(3), 311–329 (1960).

[7] S. O. Sadjadi and J. H. L. Hansen, *Hilbert envelope based features for robust speaker identification under reverberant mismatched conditions*, Proc. IEEE ICASSP, 5448–5451 (2011).

[8] G. Tzanetakis, G. Essl, and P. Cook, *Audio analysis using the discrete wavelet transform*, Proc. WSES Int. Conf. Acoustics & Music Theory & Applications, Skiathos, Greece (2001).

[9] S. B. Davis and P. Mermelstein, *Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences*, IEEE Trans. Acoust. Speech Signal Process. **28**(4), 357–366 (1980).