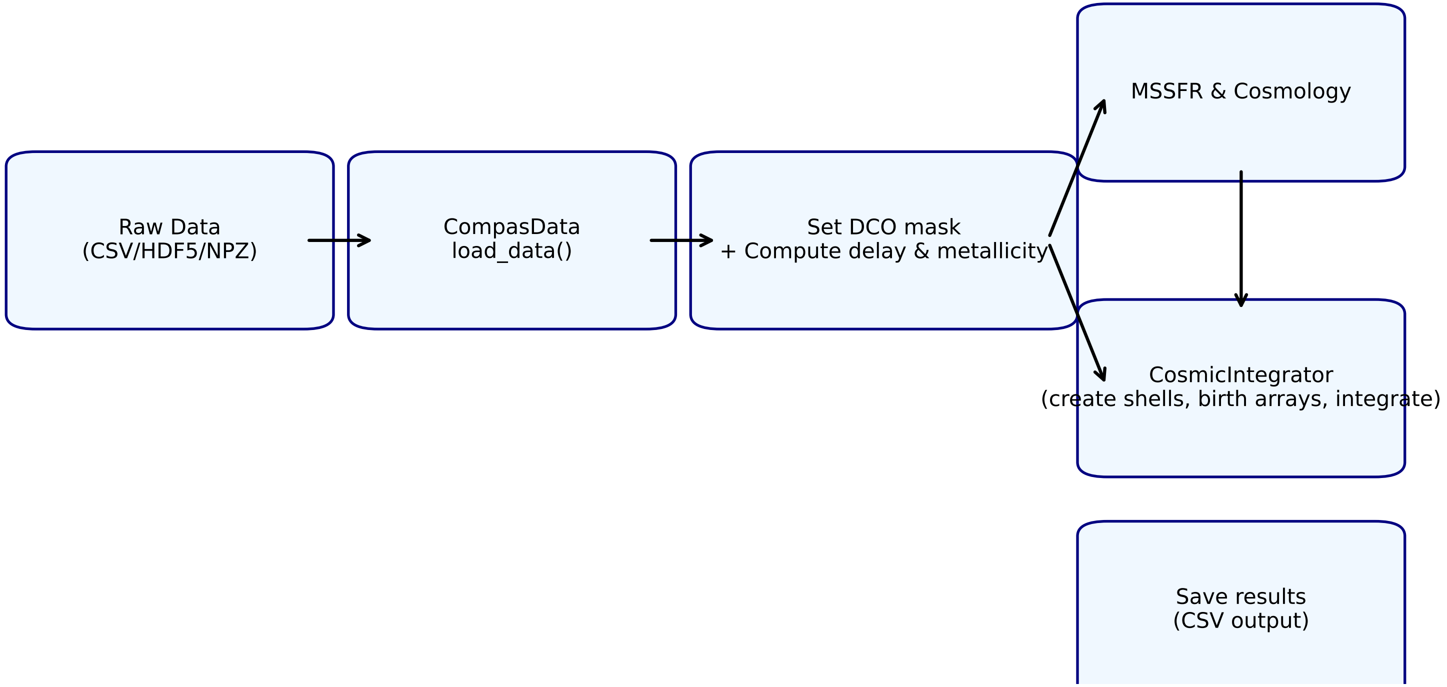
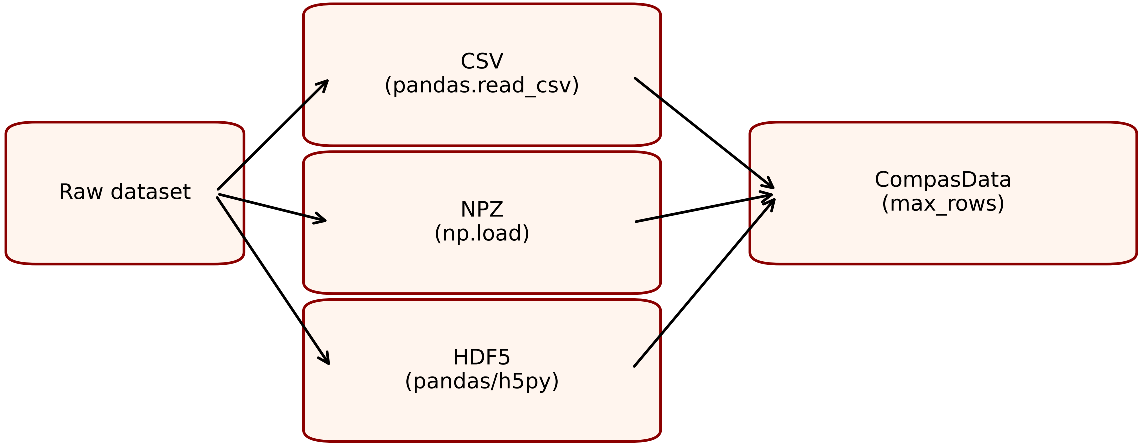

Raw Data
(CSV/HDF5/NPZ)
CompasData
load_data()
Set DCO mask
+ Compute delay & metallicity
MSSFR & Cosmology
CosmicIntegrator
(create shells, birth arrays, integrate)
Save results
(CSV output)


Raw dataset
CSV
(pandas.read_csv)
NPZ
(np.load)
HDF5
(pandas/h5py)
CompasData
(max_rows)

Overview of `cosmic_integration_dasein` ## Core functionality The `cosmic_integration_dasein` package provides a lightweight pipeline for convolving binary-population synthesis catalogues with cosmic star-formation histories to estimate the merger and detection rates of double compact objects (DCOs). Its architecture comprises a small set of classes that correspond to the stages of the astrophysical rate calculation: ### `CompasData` *Acts as a container for a population synthesis catalogue.* It accepts input files in CSV, pickle (`.pkl`/`.pickle`) or NumPy archive (`.npz`) formats and, in the updated version, can read HDF5 files when `pandas` or `h5py`/`PyTables` is available. The class validates that the expected columns (masses, stellar types, flags, metallicity and either formation/merger times or delay times) are present. Upon loading it computes the number of systems and stores the data frame internally. Important methods include: | Method | Purpose | |------------------------------|--| | `load_data()` | Read the input file into a DataFrame, automatically computing the delay time if formation and merger times are provided. The updated implementation introduces a `max_rows` parameter that limits the number of rows loaded, providing a simple form of chunked reading inspired by the **Synthetic Stellar Pop Convolve (SSPC)** project.| | `set_dco_mask()` | Construct boolean masks for different classes of double compact objects. Users can select BBH, BHNS or BNS populations and apply common-envelope and immediate Roche–lobe overflow filters. | | `compute_delay_times()` | Ensure that a `delay_time` column exists; if not present, it is calculated as `merger_time` – `formation_time`. | | `compute_metallicity_grid()` | Build a metallicity grid and compute the total mass evolved per metallicity bin using a Kroupa initial mass function. This method populates the `metallicity_grid` and `total_mass_evolved_per_z`. | `CompasDataCSV` and `CompasDataHDF5` subclass `CompasData` to provide specialised loading logic for CSV and HDF5 files, respectively. The `max_rows` attribute is propagated so that very large datasets can be sampled during testing without exhausting memory. ### `MSSFR` The *metallicity-specific star-formation rate* class provides a simple model for the cosmic star-formation history and the metallicity distribution of star-forming gas. By default it implements the Madau-Dickinson (2014) star-formation rate and a log-normal metallicity distribution, but users may supply their own metallicity grid to override the default. The key method `mssfr_for_bin(metallicity, redshift)` returns the star-formation rate for a given metallicity and redshift. In the revised version the method is vectorised over arrays of metallicities and redshifts, making it compatible with array-based operations in `CosmicIntegrator`. ### `Cosmology` This small class wraps basic cosmological calculations without relying on external packages. It provides comoving volume elements, lookback times and comoving distances for a flat ΛCDM universe given the Hubble constant and density parameters. The function `get_cosmology()` constructs a `Cosmology` instance from user-provided parameters or defaults to Planck-like values. These quantities are used when defining redshift shells and converting between lookback times and redshifts. ### `CosmicIntegrator` The central class orchestrates the integration over cosmic time. It takes a populated `CompasData` instance, an `MSSFR` model and cosmological parameters and then computes the intrinsic and observed merger rates. The workflow is: 1. **Define redshift shells** with `create_redshift_shells()`, which partitions the redshift range into bins, computes their comoving volumes (in Gpc^3) and luminosity distances. 2. **Compute birth arrays** with `compute_birth_arrays()`, which combines delay times with shell lookback times to find formation times and then inverts lookback time to birth redshift. Systems whose delay time pushes their formation before the Big Bang are marked with a sentinel redshift of –1. 3. **Integrate rates** with `integrate()`, which loops over redshift shells (in parallel if `n_workers > 1`) and sums contributions from all systems selected by the DCO mask. Each shell's intrinsic rate is computed by summing the

formation-rate weights divided by the survey volume, while the observed rate applies a simple detection probability (a placeholder using a signal-to-noise ratio threshold). 4. **Save results** using `save_results(filename)` to write a CSV containing redshift bin centres and the intrinsic and observed merger rates. Parallelisation is achieved using `ProcessPoolExecutor`, enabling independent redshift bins to be processed concurrently. The integrator is therefore able to scale to large catalogues provided the input dataset can be loaded into memory. ### `selection_effects` This module provides a basic detection probability model via `detection_probability(mass1, mass2, redshift, snr_threshold=8.0)`. It uses a simple scaling of the source redshifted chirp mass and ignores orientation and detector noise variation. Although simplistic, it illustrates where detection effects enter the cosmic integration. In a production environment this function could be replaced with a detailed detector sensitivity model or the detection matrix tools available in the optional `binned` subpackage. ## Optional modules and subpackages ### `binned` subpackage The `binned` directory contains additional tools for computing merger rates and signal-to-noise ratios on a fixed grid in primary and secondary mass and redshift. These modules require optional dependencies (e.g. `astropy`, `cupy` for GPU acceleration) and are not imported by default. Notable components include: | Module | Description and use case | |------------------------------|---| | `binary_population.py` | Defines a class representing a binned population of compact binaries on a two-dimensional mass grid. Provides methods to populate the grid from synthetic catalogues and to marginalise over one dimension. | | `conversions.py` | Contains utility functions to convert between different parameterisations (e.g. chirp mass and mass ratio). | | `cosmological_model.py` | Computes comoving volumes and horizon distances using astropy's cosmology and passes them to SNR integrators. | | `detection_matrix.py` | Implements a matrix of detection probabilities across mass and redshift bins. This is useful for fast evaluation of detection rates once the matrix has been precomputed. | | `snr_grid.py` | Constructs grids of signal-to-noise ratios for different detector sensitivities and uses them to determine which systems exceed a detection threshold. | | `stellar_type.py` | Defines numerical codes for different stellar types (BH, NS, etc.) and conversion functions. | | `gpu_utils.py` | Provides helper functions to detect and initialise CUDA devices. These routines allow grid computations to be offloaded to a GPU when available. | The optional modules are intended for more advanced analyses where histogram-based (ensemble) convolution or GPU acceleration is desired. They mirror some of the functionality found in the **Synthetic Stellar Pop Convolve** project, such as binned convolution and detection matrices. However, they require additional dependencies and are therefore disabled unless the user explicitly imports them. ### `frame_generator.py` This module offers a basic interface to generate waveform frames for gravitational-wave signals, though the current implementation is a placeholder. In a full pipeline it could be replaced by calls to waveform models (e.g. from `lalsimulation`) and used to compute detection probabilities more accurately. ### `metallicity_grid.py` Contains helper functions to compute the total mass evolved per metallicity bin under different initial mass functions. It implements a Kroupa IMF and integrates it analytically between lower and upper mass limits. Users can override the mass range and binary fraction on the `CompasData` instance before calling `compute_metallicity_grid()` to explore different population synthesis assumptions. ### `mssfr.py` Defines the `MSSFR` class described above along with helper functions for generating metallicity grids and evaluating SFR histories. It supports vectorised inputs and can be extended to include alternative SFR prescriptions. ## Integrating ideas from *Synthetic Stellar Pop Convolve* The **syntheticstellarpopconvolve** project is a comprehensive framework for convolving stellar population synthesis data with star-formation histories. Although much more feature rich than `cosmic_integration_dasein`, several concepts can be adopted: * **Chunked data access** – SSPC supports reading slices of HDF5 tables to process datasets that do not fit into memory. The introduction of a `max_rows` attribute to `CompasData` and its subclasses implements a simple form of this idea, allowing users to load just the first `n` systems from very large catalogues for debugging and testing. Extending this further to iterate over chunks and accumulate the rate calculation incrementally would improve scalability. * **Clear data–column dictionaries** – SSPC uses a dictionary to map required quantities to input columns and supports unit conversions. Adopting a similar pattern in the future would decouple the `CompasData` class from rigid column names and allow custom population synthesis outputs to be used without renaming columns. * **Post-convolution processing hooks** – the SSPC pipeline allows arbitrary functions to be applied after each convolution step (e.g. reweighting by detection probability). Integrating a callback mechanism into `CosmicIntegrator.integrate()` would enable users to implement custom selection effects without modifying the core code. ## Summary `cosmic_integration_dasein` provides a compact yet extensible framework for computing the cosmic merger and detection rates of double compact objects from population synthesis data. It consists of a handful of well-defined classes (`CompasData`, `MSSFR`, `Cosmology` and `CosmicIntegrator`) that mirror the logical steps of the rate calculation. The optional `binned` subpackage and other modules offer additional functionality for advanced analyses but are not imported unless explicitly used. By incorporating features inspired by the **Synthetic Stellar Pop Convolve** project—such as chunked data loading—the revised codebase remains lightweight while being easier to use on large datasets. Future improvements could focus on adding unit-aware data extraction, richer star-formation models and more flexible post-processing hooks.