The Open Source Land Surface Model Benchmarking System (OpenBench) Users’ Guide

CONTENTS

Part I Introduction

1. Introduction

In Earth system science and climate change research, Land Surface Models (LSMs) play a critical role. These models simulate the exchanges of energy, water, and carbon cycles between the land surface and the atmosphere, providing essential tools for understanding and predicting global climate change. However, as model complexity increases and their application scope expands, effectively evaluating and comparing the performance of different land surface models has become an increasingly prominent challenge. To address this, we developed **The Open Source Land Surface Model Benchmarking System (OpenBench)**. OpenBench is an open-source benchmarking system specifically designed for land surface model evaluation. Its goal is to provide researchers and model developers with a standardized, comprehensive, and flexible tool for validating model outputs, comparing the performance of different models, and analyzing model behavior under various conditions. By offering a unified set of evaluation standards and tools, OpenBench aims to promote continuous improvement of land surface models and collaboration within the scientific community.

OpenBench is designed with the core principles of openness, standardization, flexibility, comprehensiveness, and efficiency. **Openness** is reflected in OpenBench being an open-source project, encouraging broad participation and contributions from the scientific community to enhance the transparency and reproducibility of evaluation methods. **Standardization** ensures comparability of results across studies by providing unified evaluation metrics and data processing workflows. **Flexibility** allows users to customize the evaluation process based on specific needs and supports various types of land surface models and data formats. **Comprehensiveness** means OpenBench not only focuses on the performance of individual variables but also provides multi-variable, multi-scale evaluation capabilities. Finally, **efficiency** is achieved through parallel computing and optimized data processing workflows, enabling efficient handling of large-scale datasets.

OpenBench is a powerful and comprehensive one-stop solution for land surface model evaluation. It allows simultaneous processing of outputs from multiple models, enabling researchers to directly compare their performance under identical conditions, which is crucial for model comparison, integration, and uncertainty analysis. OpenBench supports the evaluation of numerous land surface process variables, covering ecosystem carbon cycles, hydrological cycles, energy balance, and biophysical parameters, with a comprehensive set of metrics for each variable, such as gross primary productivity, evapotranspiration, and soil moisture, to thoroughly assess model performance. Additionally, OpenBench provides basic statistical metrics and advanced scoring methods, such as Kling-Gupta Efficiency and Nash-Sutcliffe Efficiency, and supports time series analysis and spatial pattern evaluation to deeply analyze model performance across different spatiotemporal scales. OpenBench enables evaluations across temporal scales from hourly to interannual, spatial scales from site to global, and various ecosystem types, allowing users to comprehensively understand model performance under diverse conditions. To ensure the reliability of evaluation results, OpenBench integrates features like automatic data format conversion, spatiotemporal interpolation, missing data handling, and outlier detection to guarantee the quality and consistency of input data. Furthermore, OpenBench offers rich visualization tools, including time series plots, spatial distribution maps, and Taylor diagrams, which aid in result interpretation and generate high-quality figures for reports and publications. Lastly, OpenBench implements parallel computing, allowing users to configure the number of computing cores based on hardware resources, significantly improving processing speed for large datasets.

OpenBench has a wide range of applications in land surface model research and applications. It helps model developers validate newly developed model modules, assess improvements in parameterization schemes, and identify model strengths and weaknesses, thereby advancing model development. OpenBench is also suitable for multi-model comparison studies, such as climate change impact assessments, ensuring result comparability, revealing differences between models, and supporting various types of comparison analyses, making it an ideal tool for researchers. Additionally, OpenBench can be integrated into model calibration and parameter optimization workflows, providing objective functions for optimization algorithms, evaluating model performance under different parameter settings, and analyzing parameter sensitivity to help researchers identify optimal model parameters. In data assimilation and model-data fusion applications, OpenBench can evaluate model performance before and after assimilation, provide error statistics for assimilation algorithms, and verify the spatiotemporal consistency of assimilation results, ultimately improving data assimilation outcomes. OpenBench is also valuable for climate change impact assessments, evaluating model performance in historical periods, analyzing model consistency under future scenarios, and quantifying prediction uncertainties to provide reliable support for climate change impact studies. In summary, OpenBench offers comprehensive evaluation and analysis tools for land surface model research, fostering model development and application while providing robust support for weather and climate-related studies.

Part II   Quick Start

2. Running an Evaluation Instance

2.1 Code Structure

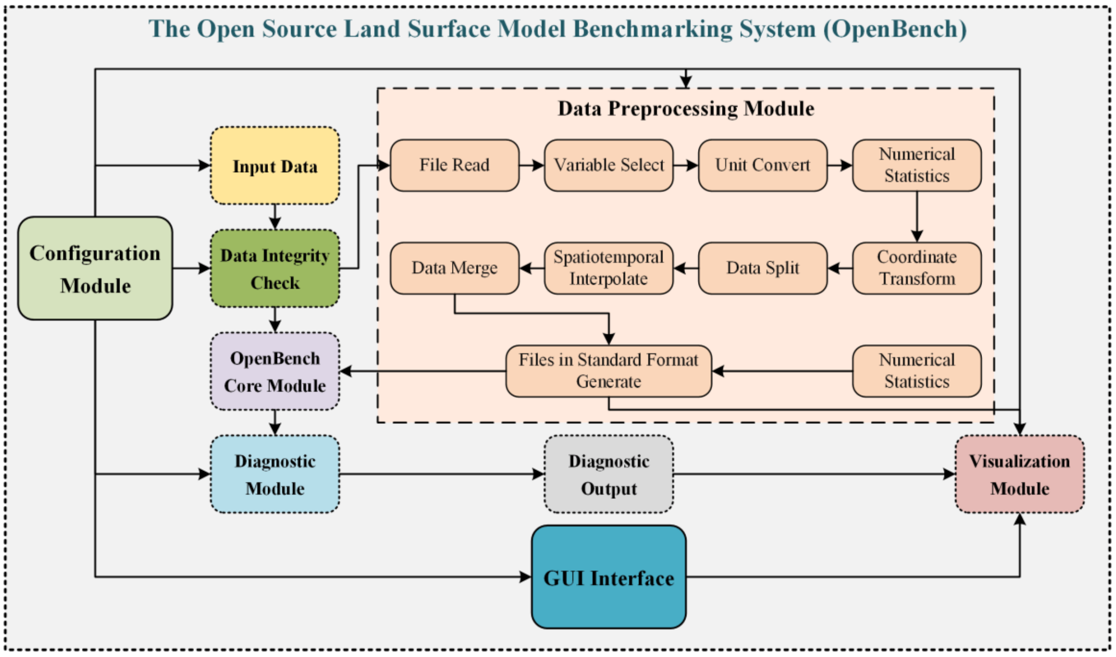

Figure 1: OpenBench Workflow Diagram

OpenBench adopts a modular design, with core components including the configuration module, data preprocessing module, diagnostic module (including evaluation module, comparison analysis module, and statistical analysis module), visualization module, and log generator. The **configuration module** reads and parses user configuration files to set evaluation parameters. The **data preprocessing module** handles model outputs and reference data, performing format conversion, alignment, and quality control. The **diagnostic module** is the core module for evaluation, comprising the **evaluation module**, the **comparison analysis module**, and the **statistical analysis module**. The **evaluation module** is designed to calculate various evaluation metrics and scoring methods. The **comparison analysis module** supports multi-model comparison and integration analysis, while the **statistical analysis module** provides advanced statistical analysis capabilities. The **visualization module** generates various charts and visual outputs, and the **log generator** compiles results into comprehensive evaluation reports.

OpenBench workflow is highly automated—users only need to prepare configuration files and input data to obtain comprehensive evaluation results. The system first reads the configuration, loads and preprocesses data, then computes the specified evaluation metrics. If the comparison function is enabled, multi-model comparison analysis is performed. Next, statistical analysis is conducted based on the configuration, followed by the generation of visualizations and evaluation logs. This modular architecture minimizes coupling between functional modules, facilitating system maintenance and scalability. During use, the configuration module is primarily manipulated.

2.2 Installation and Execution

The installation and execution of OpenBench involve four main steps: installing the evaluation system, configuring the software environment, preparing data, and running the evaluation system.

(1) Evaluation System Installation and Configuration

OpenBench currently runs on Linux systems. The current version is hosted on GitHub. Installation command:

```
git clone git@github.com:zhongwangwei/OpenBench.git
```

Assume the code is placed in the root directory **$OpenBench**.

(2) Software Environment Configuration

Software requirements for OpenBench on Linux:

```
Python (version≥3.10)
```

Main software dependencies:

```
NetCDF4 (version≥1.5.7)
xarray (version≥0.19.0)
NumPy (version≥1.21.0)
SciPy (version≥1.7.0)
Matplotlib (version≥3.4.0)
Pandas (version≥1.3.0)
joblib (version≥1.1.0)
Dask (version≥2022.1.0)
Cartopy (version≥0.20.0)
flox (version≥0.5.0)
```

The following file provides a typical software configuration for the environment, located at **$OpenBench/requirement.yml**:

```yaml
name: openbench
channels:
  - conda-forge
  - defaults
channel_priority: strict
dependencies:
  - python>=3.10
  - netCDF4>=1.5.7
  - xarray>=0.19.0
  - numpy>=1.21.0
  - scipy>=1.7.0
  - matplotlib>=3.4.0
  - pandas>=1.3.0
  - joblib>=1.1.0
  - dask>=2022.1.0
  - cartopy>=0.20.0
  - flox>=0.5.0
```

This file configures the Python interpreter and third-party Python libraries, listing the required Python packages and their versions.

If **conda** is installed, create and activate the new environment (openbench):

```
conda env create -f $OpenBench/requirements.yml
conda activate openbench
```

Alternatively, use **pip** to install:

```
pip install -r $OpenBench/requirements.txt
```

(3) Data Preparation

Prepare the two necessary datasets for running OpenBench: simulation datasets and reference datasets.

(4) Running the Evaluation System

OpenBench configures an evaluation instance using three files:

- **$OpenBench/main.json**: Main configuration file
- **$OpenBench/sim.json**: Simulation configuration file
- **$OpenBench/ref.json**: Reference configuration file

These files can be named arbitrarily. The evaluation system provides a pre-configured example corresponding to the files: **main-Debug.json**, **sim-Debug.json**, and **ref-Debug.json**, all located in the **$OpenBench** directory. Refer to Part III for detailed configuration.

Navigate to the root directory **$OpenBench** and run the example configuration:

```
python script/openbench.py nml/main-Debug.json
```

Note that the main configuration file (e.g., **main-Debug.json**) is specified during execution, while the simulation configuration file (e.g., **sim-Debug.json**) and reference configuration file (e.g., **ref-Debug.json**) are set within the main configuration file.

Excerpt from **main-Debug.json**:

```
    "reference_nml": "./nml/ref-Debug.json", // Path to reference
                                              configuration file
    "simulation_nml": "./nml/sim-Debug.json",// Path to simulation
                                              configuration file
```

Evaluation system outputs are saved in **$OpenBench/output/**.

Part III System Configuration

For system configuration, OpenBench provides fully annotated configuration files based on the Debug example, including:

(1) Main Configuration File:

```
$OpenBench/nml/main-Debug_comment_EN.json
```

(2) Simulation Configuration File:

```
$OpenBench/nml/sim-Debug_comment_EN.json
```

(3) Reference Configuration File:

```
$OpenBench/nml/ref-Debug_comment_EN.json
```

(4) Simulation Data:

Simulation Data 1 Configuration:

```
$OpenBench/nml/user/debug/grid_case_comment_EN.json
```

Simulation Data 2 Configuration:

```
$OpenBench/nml/user/debug/station_case_comment_EN.json
```

(5) Reference Data:

Reference Data 1 Configuration:

```
$OpenBench/nml/Ref_variables_defination/Debug/GLEAM_hybrid_PLUMBER2_com
ment_EN.json
```

Reference Data 2 Configuration:

```
$OpenBench/nml/Ref_variables_defination/Debug/ILAMB_monthly_comment_EN.
json
```

Reference Data 3 Configuration:

```
$OpenBench/nml/Ref_variables_defination/Debug/PLUMBER2_comment_EN.json
```

Reference Data 4 Configuration:

```
$OpenBench/nml/Ref_variables_defination/Debug/GLEAM4.2a_monthly_comment
_EN.json
```

These files include annotated explanations for all parameter settings. Since JSON files cannot include comments, use the corresponding uncommented versions (**xxx.json**) to run the example case. The configuration sequence is shown in Figure 2.

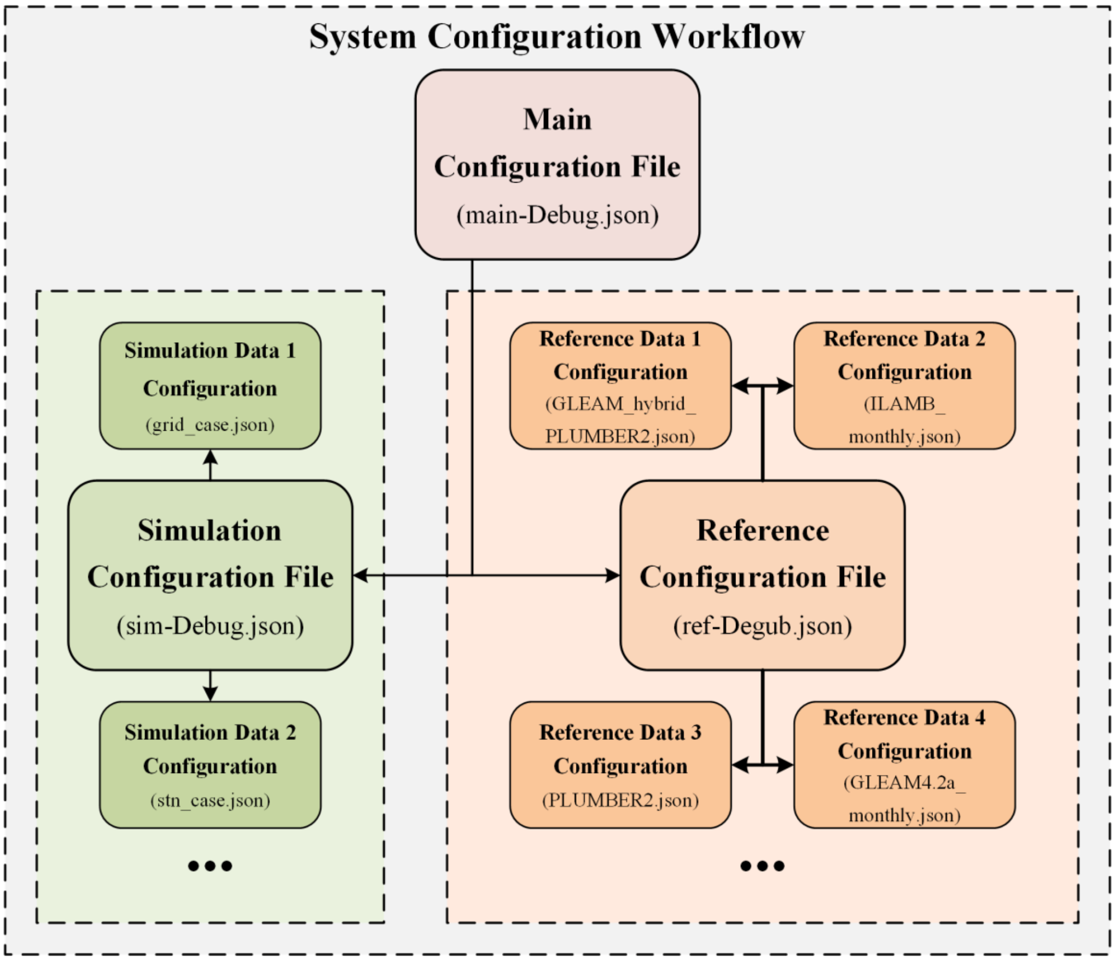
System Configuration Workflow
Main Configuration File (main-Debug.json)
Simulation Data 1 Configuration (grid_case.json)
Simulation Configuration File (sim-Debug.json)
Simulation Data 2 Configuration (stn_case.json)
•••
Reference Data 1 Configuration (GLEAM_hybrid_PLUMBER2.json)
Reference Data 2 Configuration (ILAMB_monthly.json)
Reference Configuration File (ref-Degub.json)
Reference Data 3 Configuration (PLUMBER2.json)
Reference Data 4 Configuration (GLEAM4.2a_monthly.json)
•••

Figure 2: System Configuration Workflow

3. Main Configuration File (main.json)

The main configuration file (**main.json**) is in JSON format and defines the parameters for the entire evaluation process, including basic settings, evaluation items, evaluation metrics, comparison methods, and statistical analysis.

Example: **main-Debug_comment_EN.json**

```
{
  /* Main configuration file: Defines parameters for the entire evaluation
process, including basic settings, evaluation items, evaluation metrics,
scoring methods, comparison analysis, and statistical analysis */

  /* Basic settings: Fundamental settings for the evaluation, such as
defining the case (Debug) here, where simulation configuration comes from
sim-Debug.json, reference configuration comes from ref-Debug.json, and
evaluation, comparison, and statistics modules are all enabled */
  "general": {
    "basename": "Debug",           // Evaluation case name
    "basedir": "./output",         // Output path for evaluation results
    "compare_tim_res": "month",    // Time resolution for evalution
                                      (e.g., "month", "day")
    "compare_tzone": 0.0,          // Time zone setting
    "compare_grid_res": 2.0,       // Spatial grid resolution (degrees,
                                      only work for grid)
    "syear": 2004,                 // Start year of evaluation
    "eyear": 2005,                 // End year of evaluation
    "min_year": 1.0,               // Minimum evaluation interval (less
                                      than 1 year will not be evaluated)
    "max_lat": 90.0,               // Maximum latitude of evaluation area
                                      (-90 to 90)
    "min_lat": -90.0,              // Minimum latitude of evaluation area
                                      (-90 to 90)
    "max_lon": 180.0,              // Maximum longitude of evaluation
                                      area (-180 to 180)
    "min_lon": -180.0,             // Minimum longitude of evaluation
                                      area (-180 to 180)
    "reference_nml": "./nml/ref-Debug.json", // Path to reference
                                              configuration file
    "simulation_nml": "./nml/sim-Debug.json",// Path to simulation
                                              configuration file
    "statistics_nml": "./nml/stats.json",    // Path to statistical
                                 analysis configuration file (default)
    "figure_nml": "./nml/figlib.json",       // Path to visualization
                                           configuration file (default)
    "num_cores": 48,             // Number of cores used for parallel
                                    computing
    "evaluation": true,          // Whether to enable evaluation module
    "comparison": true,          // Whether to enable comparison module
    "statistics": true,          // Whether to enable statistics module
    "debug_mode": true,          // Whether to enable debug mode (show
                                    debug information in log)
    "weight": "None",            // Whether to apply weighting ("area",
                                    "mass", "None")
    "IGBP_groupby": true,        // Whether to aggregate metics and
                                    scores by IGBP
    "PFT_groupby": true,         // Whether to aggregate metics and
                                    scores by PFT
    "unified_mask": true         // Whether to evaluate only
                                    overlapping areas when model data
                                    are all gridded
  },
  /* Evaluation items: Defines evaluable variables and their switches,
with the evaluation module enabled in basic settings, all variables set
to true here are designated as evaluation items, including
Evapotranspiration, Latent_Heat, and Sensible_Heat */
  "evaluation_items": {
    "Biomass": false,
    "Burned_Area": false,
    "Evapotranspiration": true,
    "Latent_Heat": true,
    "Sensible_Heat": true,
    /* … */

  },
  /* Evaluation metrics: Defines evaluation metrics and their switches,
with the evaluation module enabled in basic settings, all evaluation items
will calculate the metrics set to true below, including RMSE, correlation,
and KGESS */
  "metrics": {
    "RMSE": true,
    "correlation": true,
    "KGESS": true,
    "kappa_coeff": false,
    "rv": false,
    /* … */
  },
  /* Scoring methods: Defines scoring methods and their switches, with
the evaluation module enabled in basic settings, all evaluation items
will calculate the scoring methods set to true below, including
Overall_Score */
  "scores": {
    "nBiasScore": false,
    "nRMSEScore": false,
    "nPhaseScore": false,
    "nIavScore": false,
    "nSpatialScore": false,
    "Overall_Score": true,
    "The_Ideal_Point_score": false
  },
  /* Comparison analysis: Defines comparison methods used for multi-model
comparison, with the comparison module enabled in basic settings, all
evaluation items' metrics and scoring methods will use the comparison
methods set to true below, including HeatMap, Kernel_Density_Estimate,
Single_Model_Performance_Index, Ridgeline_Plot, and Standard_Deviation
*/
  "comparisons": {
    "HeatMap": true,
    "Taylor_Diagram": false,
    "Target_Diagram": false,
    "Kernel_Density_Estimate": true,
    "Whisker_Plot": false,
    "Parallel_Coordinates": false,
    "Portrait_Plot_seasonal": false,
    "Single_Model_Performance_Index": true,
    "Relative_Score": false,
    "Ridgeline_Plot": true,
    "Diff_Plot": false,
    "Mean": false,
    "Median": false,
    "Min": false,
    "Max": false,
    "Sum": false,
    "Mann_Kendall_Trend_Test": false,
    "Correlation": false,
    "Standard_Deviation": true,
    "Functional_Response": false
  },
  /* Statistical analysis: Defines statistical analysis methods, with the
statistics module enabled in basic settings, all evaluation items' metrics
and  scoring  methods  will  use  the  statistical  methods  set  to  true
below, although  the  statistics  module  is  enabled  in  this  case,  no
statistical methods are specified */
  "statistics": {
    "Z_Score": false,
    "Hellinger_Distance": false,
    "Three_Cornered_Hat": false,
    "Partial_Least_Squares_Regression": false,
    "False_Discovery_Rate": false,
    "ANOVA": false
  }
}
```

4. Simulation Configuration File (sim.json)

The Simulation configuration file (**sim.json**) specifies critical information such as the location, format, and variable names of the model output data, serving as a key component for the system to accurately read and process model data. Consistent with the main configuration file, this file is written in JSON format.

Example: **sim-Debug_comment_EN.json**

```
{
  /* Simulation configuration file: Defines the name and location of simulation output data, specific data formats, variable names, etc., should be modified in the corresponding simulation data configuration file, such as modifying grid_case.json and stn_case.json here */

  /* Basic settings: Defines the simulation data, corresponding to the model simulation data sources for evaluation items, only evaluation items enabled in the main configuration file require model simulation data sources, such as two simulation data cases here: a regional simulation case (grid_case) and a single-point simulation case (stn_case) */
  "general": {
    "Evapotranspiration_sim_source": [
      "grid_case",
      "station_case"
    ],
    "Latent_Heat_sim_source": [
      "grid_case",
      "station_case"
    ],
    "Sensible_Heat_sim_source": [
      "grid_case",
      "station_case"
    ]
  },
  /* Data source settings: Defines the configuration files for simulation data, detailed configurations should be set according to the specified file paths */
  "def_nml": {
    "grid_case": "./nml/user/debug/grid_case.json",
    "station_case": "./nml/user/debug/station_case.json"
  }
}
```

In the "def_nml" section, the parameters "grid_case" and "station_case" represent instances of model runs, which can be arbitrarily named as long as they are consistent with the names specified in the "general" section. Each instance has its own configuration file that defines detailed information about the model output data. As shown in the previous example, two instance data configurations are provided.

Example: **grid_case_comment_EN.json**

```
{
   /* Specific simulation configuration file (grid_case): Defines the
format of a specific simulation result to be compared */
  "general": {
    "model_namelist": "./nml/Mod_variables_defination/CoLM.json", //
         Model used, here CoLM results, with various model formats pre-
         configured, refer to corresponding models
         under ./nml/Mod_variables_defination/
    "timezone": 0.0,                          // Time zone used by the
         model results
    "data_type": "grid",                      // Data type ("grid" or
         "stn")
    "data_groupby": "month",                  // Data aggregation type
         (for grid: "year", "month", "day"; for stn: "single")
    "fulllist": "",                           // Specific to stn, full
         list of data, set to "" for grid
    "tim_res": "month",                       // Time resolution
         ("year", "month", "day")
    "grid_res": 2.0,                          // Spatial resolution
         (degrees)
    "suffix": "",                             // Suffix for data name
         (here, full data name format is grid_case_hist_2004-01.nc, no
         suffix)
    "prefix": "grid_case_hist_",                // Prefix for data name
         (here, full data name format is grid_case_hist_2004-01.nc,
         prefix is grid_case_hist_)
    "syear": 2004,                              // Start year (beginning
         in January of this year)
    "eyear": 2005,                              // End year (ending in
         December of this year)
    "root_dir": "./data/simulation/debug/grid"  // Data storage path
  }
}
```

5. Reference Configuration File (ref.json)

The reference data configuration file (ref.json) specifies critical information about the reference data used for evaluating model performance, including data sources, format, and variable names. These reference data may encompass in-situ observations, satellite data, reanalysis data, or other sources. Consistent with other configuration files, this file is written in JSON format.

Example: **ref-Debug_comment_EN.json**

```
{
  /* Reference configuration file: Defines the name and location of reference data used for evaluating model performance, specific data formats, variable names, etc., should be modified in the corresponding reference data configuration file, OpenBench has pre-configured a large number of model reference data configurations, if using the corresponding reference data, the preset reference data configuration can be used directly, for details, refer to the preset reference data paths and corresponding configurations in ref.json, multiple reference data sources can be set for any evaluation item, reference data may include in-situ observations, satellite data, reanalysis data, etc. */

  /* Basic settings: Defines the reference data sources for each evaluation item, only evaluation items enabled in the main configuration file require reference data sources, the sources for different evaluation items can vary */
  "general": {
    "Evapotranspiration_ref_source": [
      "GLEAM4.2a_monthly",
      "GLEAM_hybird_PLUMBER2"
    ],
    "Latent_Heat_ref_source": [
      "ILAMB_monthly",
      "PLUMBER2"
    ],
    "Sensible_Heat_ref_source": [
      "ILAMB_monthly",
      "PLUMBER2"
    ]
  },
  /* Data source settings: Defines the specific configurations for the
reference data sources specified above, detailed configurations should be
set according to the specified file paths */
  "def_nml": {
    "GLEAM_hybird_PLUMBER2":
"./nml/Ref_variables_defination/Debug/GLEAM_hybird_PLUMBER2.json",
    "ILAMB_monthly":
"./nml/Ref_variables_defination/Debug/ILAMB_monthly.json",
    "PLUMBER2": "./nml/Ref_variables_defination/Debug/PLUMBER2.json",
    "GLEAM4.2a_monthly":
"./nml/Ref_variables_defination/Debug/GLEAM4.2a_monthly.json"
  }
}
```

Each reference dataset has its own configuration block that defines detailed information about the dataset, stored by default in the **$OpenBench/nml/Ref_variables_defination** directory. As illustrated in the previous example, four reference data configurations are provided.

Example: **ILAMB_monthly_comment_EN.json**

```
{
  /* Specific reference data configuration file (ILAMB_monthly): Defines
the format of a specific reference data, unified reference data may
include multiple evaluation items, here including Sensible_Heat and
Latent_Heat */

  /* Basic settings: Configures the format of this reference data */
  "general": {
    "root_dir": "./data/reference/debug/ILAMB", // Data storage path
    "timezone": 0.0,                           // Time zone used by the
          data
    "data_type": "grid",                       // Data type ("grid" or
          "stn")
    "data_groupby": "Single",                    // Data aggregation type
             (for grid: "year", "month", "day"; for all times aggregated
             together or stn: "single")
    "tim_res": "Month",                          // Time resolution
             ("year", "month", "day")
    "grid_res": 2.0                              // Spatial resolution
             (degrees), can be "" for stn
  },
  /*  Reference  data  variable  settings: Configures  the  format  of  all
evaluation variables in this reference data, here for grid, settings for
Sensible_Heat  and  Latent_Heat  include path  prefix,  variable  name,
variable unit, file name, start and end years, etc. */
  "Sensible_Heat": {
    "sub_dir": "Sensible_Heat/FLUXCOM",
    "varname": "sh",
    "varunit": "w m-2",
    "prefix": "sh",
    "suffix": "",
    "syear": 2004,
    "eyear": 2005
  },
  "Latent_Heat": {
    "sub_dir": "Latent_Heat/FLUXCOM",
    "varname": "le",
    "varunit": "w m-2",
    "prefix": "le",
    "suffix": "",
    "syear": 2004,
    "eyear": 2005
  }
}
```

To incorporate a new custom reference dataset, a custom variable definition file must be created. The file can be named arbitrarily, but its content should follow the structure of the **ILAMB_monthly** variable definition file shown above.

6. Output Data and Charts

The evaluation results are stored in the **$OpenBench/output/ basename** directory. For the Debug case, the output is saved in **$OpenBench/output/Debug**, including the following subdirectories:

- **log/**: Contains system logs for the OpenBench evaluation run.
- **output/comparisons/**: Stores results from comparision analyses.
- **output/data/**: Includes processed data for each simulation and reference dataset used in the evaluation. For station data, this directory also contains time series line charts.
- **output/metrics/**: Contains results of various evaluation metrics.
- **output/scores/**: Includes results of various scoring methods.
- scratch/: Pre-treatment temporary folder, such as files split by year.
- tmp/: Intermediate temporary folder, such as the resampled files.

The output directory contains corresponding result charts and data files.

Part IV Developer User Guide

7. Advanced Usage

7.1 Custom Evaluation Metrics

Users can add new evaluation metrics and scoring methods by modifying **Mod_Metrics.py** and **Mod_Scores.py**.

7.2 Parallel Computing

Parallel computing can be enabled to improve performance by setting the num_cores parameter in main.json.

7.3 Custom Data Processing

To support new data formats, users can modify **Mod_DatasetProcessing.py**.

8. Frequently Asked Questions

Q: How to add a new evaluation variable?

A: Add the new variable to the “evaluation_items” section in main.json, and ensure corresponding data paths are provided in “simulation_nml” and “reference_nml”.

Q: How do I handle data with different temporal resolutions?

A: OpenBench automatically resamples data to the temporal resolution specified in “compare_tim_res”. Ensure the original data's temporal resolution is correctly set in the configuration files.

Q: Why does the system report a Cartopy error during the first run after installing dependencies?

A: After installing Cartopy, run the evaluation system for the first time with an internet connection. If an internet connection is unavailable, follow the manual download instructions in the troubleshooting guide.

9. Troubleshooting

- If memory errors occur, try reducing the num_cores value in main.json or testing with a smaller dataset.
- Ensure all input data paths are correctly specified in the configuration files.
- Verify that the correct Python version and all required dependency libraries are properly installed.
- OpenBench requires Cartopy to download geographic data during its first run. If an internet connection is unavailable, follow the manual download procedure below:

9.1 Manual Cartopy data download procedure

(1) Determine Cartopy data directory:

```python
import cartopy
print(cartopy.config['data dir'])
# The usual path is: /home/username/.local/share/cartopy
```

(2) Download data from Natural Earth: On a machine with internet access, visit the Natural Earth website to download data (Cultural and Physical categories):

[Natural Earth » Downloads - Free vector and raster map data at 1:10m, 1:50m, and 1:110m scales]

(3) Required core files (110m resolution):

- Coastline: **ne_110m_coastline.zip**
- Country Boundaries: **ne_110m_admin_0_boundary_lines_land.zip**
- Land: **ne_110m_land.zip**
- Ocean: **ne_110m_ocean.zip**
- Lake: **ne_110m_lakes.zip**

(4) Prepare directory structure:

```
└── cartopy_data_dir/  # directory printed in the previous step
    ├── shapefiles/
    │   ├── natural_earth/
    │   │   ├── cultural/
    │   │   └── physical/
    └── raster/
        └── natural_earth/
```

(5) Manual installation steps:

```
# (Please replace the actual path):
cd /home/username/.local/share/cartopy
mkdir -p shapefiles/natural_earth/physical
mkdir -p shapefiles/natural_earth/cultural
# Unzip the downloaded file to the corresponding directory:
unzip ne_110m_coastline.zip -d shapefiles/natural_earth/physical/
unzip ne_110m_admin_0_boundary_lines_land.zip -d shapefiles/natural_eart
h/cultural/
```

(6) Force use of local data in code:

```
import cartopy
import cartopy.feature as cfeature
# Force the use of local data (avoid attempting to download):
coastline = cfeature.NaturalEarthFeature(
    category='physical',
    name='coastline',
    scale='110m',
    facecolor='none'
)
ax.add_feature(coastline, edgecolor='black')
```

(7) Verify installation:

```
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111, projection=ccrs.PlateCarree())
ax.coastlines()
plt.show()
```

(8) Additional downloads for offline environments:

- **ne_110m_graticules_*.zip** (grid for latitude and longitude)
- **ne_110m_geography_*.zip** (geographical indication)
- **ne_110m_admin_*.zip** (administrative division)

Notes:

- Ensure the file directory structure matches Cartopy's expectations.
- All unzipped .shp files should be placed directly in the **physical/cultural/** directories.
- For team use, the Cartopy data directory can be packaged and shared, then unzipped to each user's Cartopy data directory.
- The 110m resolution is suitable for global mapping. For higher precision, download 50m or 10m resolution data as needed.