ODIN: An online data interpolator for the ICON-ART atmospheric transport model using the ComIn interface

-

Bachelor Thesis

Zeno Hug, zenhug@student.ethz.ch

Handed in on August 27, 2025

Supervisors: Prof. Dr. Dominik Brunner,
Dr. Michael Steiner
Atmospheric Modelling and Remote Sensing group
Empa

Abstract

We present a flexible and efficient plugin for the ICON atmospheric model that enables the extraction of model variables at user-defined observation locations during runtime. The plugin supports a wide range of observation types, including stationary monitoring sites, mobile trajectories (e.g., aircraft or drones) and observations from satellites such as TROPOMI with varying pixel sizes and locations. It applies both horizontal and vertical interpolation, and can perform postprocessing steps such as the application of averaging kernels for satellite observations.

All configuration is handled via a YAML interface, which allows users to specify sampling locations, variables, and processing parameters without modifying the source code. The plugin is fully integrated with ICON's ComIn interface and scales efficiently in parallel environments using MPI. Output is written directly to structured NetCDF files for downstream analysis, greenhouse gas inversion systems, or data assimilation. The preliminary results show negligible impact on the performance of ICON.

This plugin thus provides a reproducible and modular framework for in-simulation data extraction. Key advantages include sampling exactly at observation times without temporal interpolation, eliminating the need for external postprocessing, reducing data storage requirements, and supporting multiple observation types in a single simulation. A testcase is provided to demonstrate its usage and facilitate adoption.

Contents

List of Figures 5

List of Tables 5

1 Introduction 6

1.1 ICON ART . . . 6

1.2 Traditional interpolation of station and satellite datapoints . . . 6

1.3 Objectives of this work . . . 7

2 Methods 9

2.1 ICON ComIn: A Modular Interface for ICON Extensions . . . 9

2.2 Compiling . . . 10

2.3 Types of Data . . . 11

2.3.1 Time-averaged point data . . . 11

2.3.2 Instantaneous point data . . . 11

2.3.3 Vertical profile data . . . 11

2.3.4 CIF use case . . . 12

2.3.5 Design decisions . . . 12

2.3.6 Sampling strategy flag . . . 13

2.4 Structure of the Code . . . 13

2.4.1 MPI ownership and pre-computation . . . 14

2.4.2 Run-time life-cycle . . . 14

2.4.3 Data containers . . . 15

2.4.4 Operator algebra . . . 15

2.4.5 I/O strategy . . . 15

2.4.6 Upper atmosphere extension with CAMS fields . . . 16

2.4.7 Take-away . . . 16

2.5 Interpolation Methods . . . 16

2.5.1 Horizontal Interpolation . . . 16

2.5.1.1 Point-based interpolation (stations, satellite CIF) . . . 16

2.5.1.2 Area-based interpolation (satellite CH_4) . . . 17

2.5.2 Vertical Interpolation . . . 17

2.5.2.1 Points outside the lower or upper model boundaries 18
2.5.3 Application of Averaging Kernels 18
2.5.3.1 Pre-computation and reuse 18
2.6 Efficiency 19
2.6.1 MPI 19
2.6.2 Precomputation 19
2.6.3 Vectorization 19
2.7 Modularity 20
2.7.1 Independent satellite and station modules 20
2.7.2 Support for any 3D ICON variable 20
2.7.3 Separation of configuration and code 20
2.7.4 Extensible output structure 20
2.8 Interface 21
2.8.1 YAML configuration file 21
2.8.2 Runtime flexibility 21
2.8.3 Minimal ComIn surface 22
2.9 Testcase 22
2.9.1 Contents 22
2.9.2 Getting started 22
2.9.3 Purpose 22

3 Results and Discussion **24**
3.1 Monitoring station data 24
3.2 Mobile sampling: aircraft-style flight 25
3.3 Satellite-style data: TROPOMI CH_4 26
3.4 Comparison with traditional postprocessing 29
3.5 Performance benchmarking 30

4 Conclusions **33**

5 Data and Code Availability **35**

A Appendix **36**

B Statement on the Use of AI Tools 39

References 40

Bibliography 40

List of Figures

1 Schematic overview of ComIn entry points within the ICON control flow. Green boxes indicate entry points; dark and light grey areas represent fast and slow physics respectively. Adapted from Hartung et al. [2025]. 10

2 Figure showing the principle of a satellite pixel on an ICON grid. 17

3 Time series of simulated CH_4 mixing ratios at a fictive monitoring station in Romania, extracted with one-hour averaging. The plugin produces continuous output aligned with the defined station configuration. 24

4 Time series of simulated near-surface temperature at the same station, averaged over one-hour intervals. The consistent coverage demonstrates reliable performance for multiple variables. 25

5 Comparison of observed CH_4 mixing ratios from an aircraft flight over Romania (black) with ICON model output extracted along the same trajectory using the plugin (red). The results demonstrate the ability to generate observation-equivalent time series for direct validation against in situ measurements. 26

6 Comparison of column-averaged CH_4 (XCH_4) over Romania in January 2019: TROPOMI retrievals (top), ICON model output processed with the plugin (middle) and the difference (bottom). The plugin applies vertical interpolation, CAMS-based profile extension, and averaging kernels to produce observation-equivalent satellite data. 27

7 Differences in column-averaged CH_4 (XCH_4) between the plugin-based sampling and the traditional offline interpolation workflow, with TROPOMI retrievals shown for reference. The plugin avoids errors introduced by temporal interpolation between model outputs. 29

8 Scatterplot comparing plugin-derived and traditionally processed XCH_4 values. While no systematic bias is observed, the spread indicates improved temporal fidelity from in-plugin sampling (RMSE of around 1 ppb). 30

9 Mean runtimes of ICON simulations with and without the plugin enabled. Error bars indicate the standard deviation (Note: y-axis truncated to highlight small differences). 31

List of Tables

I Runtime of ICON simulations with and without the plugin enabled. Times are reported in seconds. 31

1 Introduction

1.1 ICON ART

The ICOsahedral Non-hydrostatic model (ICON) is a state-of-the-art numerical model for atmospheric simulations across a wide range of spatial and temporal scales. It was developed by the Deutscher Wetterdienst (DWD), the Max Planck Institute for Meteorology (MPI-M), and partner institutions. ICON is applied both in operational numerical weather prediction (NWP) and in climate research. Its unstructured icosahedral grid and non-hydrostatic dynamical core allow for seamless applications from global climate modeling to limited-area high-resolution forecasts [Rieger et al., 2015].

Building upon the ICON framework, the ART (Aerosols and Reactive Trace gases) model extends ICON to include online tracer transport and atmospheric composition processes. ICON-ART couples meteorology and atmospheric chemistry within a single simulation framework, enabling comprehensive studies of aerosols, chemical tracers, and their feedbacks with dynamics and radiation, as well as supporting applications such as greenhouse gas (GHG) inversions [Thanwerdas et al., 2025; Steiner et al., 2024]. Its design supports both global and regional domains, offering flexibility in applications ranging from air quality forecasting to climate-chemistry interactions [Rieger et al., 2015; Schröter et al., 2018; Weimer et al., 2017].

The ICON-ART model system has evolved significantly over recent years. While the initial version ICON-ART 1.0 introduced the infrastructure for online-coupled composition modeling [Rieger et al., 2015], ICON-ART 2.1 further generalised the tracer framework and improved modularity, making it suitable for a broader range of applications in both research and operational contexts [Schröter et al., 2018]. Today, ICON-ART is actively used for volcanic ash, dust and pollen forecasting at DWD and in various research projects investigating the interactions between atmospheric constituents and meteorology.

1.2 Traditional interpolation of station and satellite datapoints

The traditional way of saving the results of a simulation is to write out the prognostic variables at discrete time intervals (e.g., every 15 minutes or hour). ICON can write output on a regular latitude-longitude grid or on its native unstructured grid. To compare these gridded outputs with point-based observations, such as surface monitoring stations, aircraft, ships, or satellites, interpolation in time and space is required.

Temporal interpolation. Given two consecutive model output times t_0 and t_1 bracketing an observation time t_{obs}, model fields are interpolated linearly in time. This simple approach, although assuming smooth temporal evolution, has been shown to perform well for variables such as wind speed in data-denied regions, and often performs comparably to higher-order alternatives in many atmospheric applications [Lönnqvist et al., 2009].

Vertical interpolation. Once the time-interpolated profile is computed, vertical interpolation is required to sample model fields at the altitude of the observation (e.g. pressure levels in radiosondes, flight altitudes for aircraft). If the model output is on geometric height levels, linear interpolation is typically performed directly in height. For pressure-coordinate output, a common approach is to interpolate linearly in $\log(p)$, which better represents the approximately

exponential decrease of pressure with altitude. Given the high vertical resolution in modern NWP models, both methods generally yield sufficient accuracy.

Horizontal interpolation. After time and vertical interpolation, horizontal interpolation is applied using the surrounding grid points in the horizontal plane. A commonly used method is inverse-distance weighting (IDW), where the interpolated value $\hat{u}(x_0)$ at the target location x_0 is computed as a weighted sum of the N nearest grid values:

$$\hat{u}(x_0) = \sum_{i=1}^{N} w_i u(x_i), \quad \text{with} \quad w_i = \frac{1/d_i}{\sum_{j=1}^{N} 1/d_j},$$

where $d_i = d(x_0, x_i)$ is the horizontal distance between the target point and the i-th neighbour. This formulation ensures that the weights w_i are normalized ($\sum_i w_i = 1$) and gives higher influence to closer points. IDW is straightforward to implement and widely used for grid-to-point interpolation in atmospheric and environmental data processing [Shepard, 1968; Rijaf et al., 2020].

In summary, the typical workflow for sampling ICON output at an arbitrary observation point $(x_0, z_0, t_{\text{obs}})$ consists of:

1. Temporal linear interpolation between two output times;
2. Vertical linear interpolation between adjacent model levels;
3. Horizontal interpolation using inverse-distance weighting over neighbouring grid cells.

These steps are traditionally performed in a post-processing step on the model output, which often requires large amounts of I/O operations. Especially in the case of satellite observations, such post-processing may add substantially to the overall computational cost. In this study, we demonstrate how such sampling can be performed directly within ICON using the ComIn interface, thereby removing the need for post-processing.

1.3 Objectives of this work

While the traditional interpolation workflow described above is widely used, it comes with the following disadvantages:

1. **Temporal inaccuracy vs. storage cost:** The model state is only available at discrete output times, so interpolation to observation times between these steps inevitably introduces uncertainty. Higher temporal accuracy can be achieved by writing output more frequently, but this drastically increases data volume and I/O load.
2. **Inefficient storage and data handling:** The traditional approach usually requires writing complete 3D model fields to disk, even if only a small subset of variables or locations is ultimately needed, resulting in large storage demands and significant post-processing overhead.

These two points are, in effect, a trade-off: achieving high temporal accuracy traditionally comes at the cost of massive output and processing requirements. By integrating the sampling process directly into ICON via the ComIn interface, this work removes that trade-off—enabling precise temporal sampling at the locations of interest without the need for dense output in time, and thus avoiding the associated storage and I/O penalties.

2 Methods

2.1 ICON ComIn: A Modular Interface for ICON Extensions

In this work, we use the Community Interface (ComIn) for ICON, a lightweight and flexible interface layer designed to connect ICON with external components such as diagnostics, submodels, or Earth system modules [Hartung et al., 2025]. Its modular design enables extensions to be developed and maintained independently of ICON itself, ensuring compatibility across versions and reducing maintenance effort.

ComIn is built around two core elements:

- Callback library: It defines a set of entry points in the ICON control flow where plugin functions can be executed.

- Adapter library: It provides structured access to ICON's internal state and metadata, including prognostic fields, grid geometry, and MPI domain decomposition. It also supports dynamic registration of new variables from plugins, making them available within ICON's memory and output routines.

Plugins are compiled separately as shared libraries and loaded dynamically at runtime. This decouples plugin development from ICON's core codebase and ensures minimal effort when upgrading ICON versions. Plugins can be written in Fortran, C/C++, or Python, with the Python adapter providing efficient NumPy-based access to ICON arrays, particularly useful for diagnostics, prototyping, and machine learning applications.

A key feature of ComIn is its system of entry points, placed consistently before and after major physical parameterisations such as turbulence, convection, or radiation. When ICON reaches an entry point, all registered plugin routines for that point are executed synchronously in a blocking fashion, allowing them to observe, modify, or replace parts of the model state. Figure 1 illustrates the placement of these entry points within ICON's time loop.

Importantly, ComIn enables multiple plugins to run simultaneously in a single simulation, supporting combinations such as chemistry–diagnostics setups or land-surface modules with real-time output control.

The data exchange between ICON and plugins occurs via shared memory pointers, ensuring minimal overhead. This design philosophy, tight yet non-intrusive integration, allows synchronous execution, efficient MPI parallelisation, and clean modular development, making ComIn an ideal platform for scientific extensions such as the tracking and interpolation plugin developed in this work.

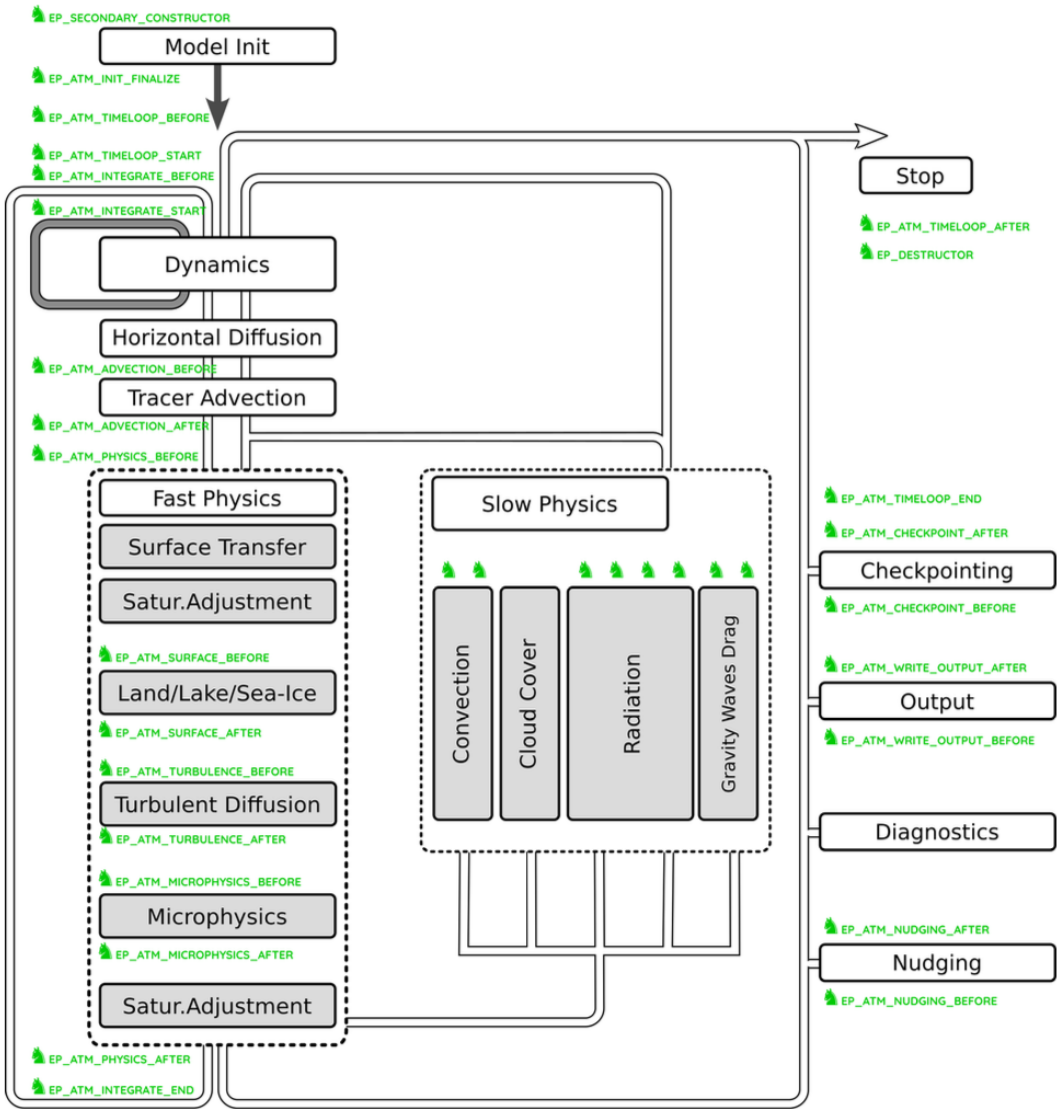

Figure 1: Schematic overview of ComIn entry points within the ICON control flow. Green boxes indicate entry points; dark and light grey areas represent fast and slow physics respectively. Adapted from Hartung et al. [2025].

2.2 Compiling

To run the plugin within ICON, the model must be compiled with ComIn support enabled. This involves two steps:

1. Compile ICON with ComIn enabled. This activates the Python plugin interface within ICON's build system.
2. Build the ComIn Python adapter. This compiles the adapter that links ICON with the Python plugin.

The plugin source code is publicly available (see Section 5), with the core implementation located in the plugin directory. The main executable script is tracker_anything.py, supported by additional modules for monitoring stations and satellite sampling.

A complete set of build instructions, including environment configuration and compilation commands for the Alps Eiger machine at the Swiss National Supercomputing Centre CSCS, is provided in the accompanying notes.txt file in the documentation directory of the repository. These instructions ensure reproducibility of the setup and allow users to adapt the build process to different computing environments.

This modular compilation approach ensures that the plugin remains decoupled from ICON's main source code, simplifying maintenance and enabling straightforward updates to new ICON releases.

2.3 Types of Data

The plugin supports several types of observational data, distinguished by their temporal resolution and spatial structure. These cover typical use cases in atmospheric science, ranging from fixed surface monitoring stations to satellite retrievals.

2.3.1 Time-averaged point data

This data type emulates stationary monitoring stations, such as those in the Integrated Carbon Observation System (ICOS) and national meteorological station networks (e.g., SwissMetNet operated by MeteoSchweiz), that record averages of atmospheric or meteorological quantities over defined time intervals, typically minutes to hours [Heiskanen et al., 2022]. ICOS is a European in situ observation and information system that currently includes over 140 standardized stations across multiple countries, designed to monitor greenhouse gas concentrations and fluxes using consistent methods and quality-controlled procedures. Likewise, meteorological networks routinely provide minute- to hourly-averaged variables such as temperature, wind, humidity, pressure, and precipitation using harmonized instrumentation and quality control. The observation point is assumed to remain spatially fixed during the averaging period, allowing the construction of continuous time series at that location.

2.3.2 Instantaneous point data

In contrast, instantaneous sampling captures the model field at an exact point in space and time. This mode is particularly relevant for mobile platforms such as aircraft, drones, or research vehicles. No temporal averaging is applied, meaning that the temporal resolution corresponds to the dynamical timestep of the model, enabling high-resolution reconstructions of trajectories or direct comparison with time-resolved measurements.

2.3.3 Vertical profile data

Some instruments, notably satellites, are sensitive to the entire vertical column of the atmosphere and report measurements in units such as column-average mole fractions. For these cases, the plugin extracts full vertical profiles of a given variable (e.g., CH_4) at a specified location and time. Two workflows are supported:

- direct output of raw vertical profiles for postprocessing outside ICON, and
- in-plugin postprocessing, including application of averaging kernels (AKs) and stratospheric profile extension required when ICON does not sufficiently cover the upper atmosphere.

2.3.4 CIF use case

An important application is integration with the inversion framework CIF, a flexible, open-source system designed specifically for greenhouse-gas (GHG) inversions across global and regional scales [Berchet et al., 2021]. ICON-ART can be used as the atmospheric transport model within CIF together with ensemble-based inversion methods: for example, Thanwerdas et al. [2025] demonstrate its use with the ensemble square-root filter (EnSRF) formulation inside CIF for GHG inversions. For satellite data, CIF applies its own postprocessing (e.g., averaging-kernel application). The plugin therefore provides raw vertical profiles, horizontally and vertically interpolated, but without further transformation, ensuring a clean separation of tasks and avoiding duplication of logic.

For monitoring stations, we distinguish between a dedicated CIF mode and a more general non-CIF mode. In non-CIF mode, all variables defined in the vars dictionary are tracked simultaneously for every station. In CIF mode, by contrast, a single variable must be specified for each sampling event, and only that variable is recorded. This design enables targeted sampling for inversion workflows while non-CIF mode remains available for broader multi-variable monitoring.

2.3.5 Design decisions

Based on the requirement of supporting both point and profile observations, the plugin implements two modular pathways:

1. **Point sampler module:** extracts model values at a specific 3D location. Supports both instantaneous sampling and temporal averaging over configurable time windows. Used for monitoring stations and mobile in-situ observations.
2. **Profile sampler module:** extracts full vertical profiles at a given horizontal location and time. Two sub-modes are currently supported:
   - Raw profile extraction of any user-defined variable,
   - CH_4-specific workflow: profile extension using output from the global atmospheric composition model CAMS (Copernicus Atmosphere Monitoring Service), application of averaging kernels, and output of a single satellite-equivalent column value.

This modular design allows the same core infrastructure to support a variety of scientific workflows, from simple station averaging to satellite emulation.

2.3.6 Sampling strategy flag

For surface monitoring stations, different sampling contexts are represented by a strategy flag included in the station input file. This flag specifies both the vertical reference and whether the values are temporally averaged or instantaneous:

- 1: Lowland station, averaged over a time interval (uses sampling height above ground)
- 2: Mountain station, averaged (uses elevation above sea level)
- 3: Lowland station, instantaneous (uses sampling height)
- 4: Mountain station, instantaneous (uses elevation)

This classification ensures that ICON interprets the vertical coordinate system appropriately and reflects the measurement conditions at both near-surface and elevated sites. The different sampling for lowland and mountain stations accounts for the fact that the model topography is usually smoother than the real topography and as a result, the top of a mountain is lower in the model than in reality.

2.4 Structure of the Code

The plugin is written entirely in Python 3 and organised into a main executable script (tracker_anything.py) supported by two helper modules (satellite.py and monitoring_stations.py). Conceptually, the program is structured into four layers.

1. **Front-end callback layer**: registers functions to ICON's ComIn interface at predefined hooks (e.g., `EP_SECONDARY_CONSTRUCTOR`, `EP_ATM_INIT_FINALIZE`, `EP_ATM_TIMELOOP_START`, `EP_ATM_TIMELOOP_END`, `EP_DESTRUCTOR`).

   These callbacks provide access points for the plugin within ICON's runtime.

2. **Domain-aware data-container layer**: Uses container objects to hold data for each task family (e.g., monitoring stations, satellite CIF, satellite CH_4). This avoids reliance on large dictionaries and ensures clear ownership of arrays.

3. **Numerical core layer**: Performs the core scientific operations, including horizontal and vertical interpolation, averaging-kernel application and related bookkeeping. These routines are implemented using efficient NumPy vector operations to minimize Python overhead.

4. **I/O layer**: Handles creation and management of NetCDF files. Headers are written once at the start of the simulation, and data are appended in blockwise batches via MPI rank 0, avoiding costly header rewrites and reducing I/O overhead.

This layered structure separates ICON–ComIn interaction from numerical processing and I/O, ensuring both maintainability and performance. Each component can be modified independently, allowing the plugin to be extended with minimal changes to existing code.

2.4.1 MPI ownership and pre-computation

To ensure that each observation is processed exactly once and without duplication across MPI ranks, ownership is assigned during the EP_ATM_INIT_FINALIZE callback.

a) Cell centres (clon/clat) are converted to 3-D Cartesian coordinates; an in-memory KDTree is built.

b) For every observation we query the tree for the k nearest cells, check whether the closest cell belongs to the local sub-domain (`decomp_domain == 0`), and apply a distance threshold (`accepted_distance`) to reject orphan points.

c) Horizontal weights are inverse-distance normalised; vertical indices and weights are either derived from `hhl` or during runtime through pressure.

All resulting arrays are stored in a rank-local "*to-do*" container (`StationDataToDo`, `SatelliteDataToDo_CH4`, ...). From this moment on each rank works strictly on its own slice of observations.

2.4.2 Run-time life-cycle

The simulation loop follows a consistent four-phase life cycle, designed to balance numerical accuracy with efficient memory use.

Time-loop start (hook `EP_ATM_TIMELOOP_START`) Data filed from the global model CAMS are available at 6-hourly time intervals. Therefore, every sixth model hour, the current and the next CAMS files are opened to enable linear time interpolation.

Field gathering At `EP_ATM_TIMELOOP_END`, the plugin converts all ICON fields requested via `comin.var_get` into `numpy` arrays. This keeps the numerical core completely library-free and avoids repeated Fortran/Python crossings.

Tracking Each task family calls its dedicated tracking routine: `tracking_points`, `tracking_points_cif`, `tracking_satellite_pressures`, or `tracking_CH4_satellite`.

- build (or reuse) interpolated profiles,
- update running sums or directly compute column values,
- move finished observations from the "to-do" to the "done" container.

Batch write-out Once `time_interval_writeout` seconds have elapsed, each rank sends its "done" slice to rank 0, which appends the data blockwise to the NetCDF file. Counters are reset and memory is reused, so no large reallocations occur.

2.4.3 Data containers

To avoid overhead from large dictionaries and repeated lookups, the plugin employs lightweight container classes that expose only attributes and allow in-place masking when needed.

- `StationDataToDo` / `StationDataDone`
- `SatelliteDataToDoGeneral` / `SatelliteDataDoneGeneral`
- `SatelliteDataToDoCH4` / `SatelliteDataDoneCH4`

This separation keeps the hot numerical loops free of dictionary look-ups and allows clear ownership: every array 'lives' exactly in one container.

2.4.4 Operator algebra

User-defined variables may be linear combinations of ICON native fields. To keep the interpolation kernels agnostic, a small `operations_dict` maps the YAML keywords `plus` and `minus` to `operator.add` and `operator.sub`. Scaling factors are parsed as `floats` during configuration loading, removing all string handling from the inner loops.

2.4.5 I/O strategy

NetCDF headers (dimensions, metadata, attributes) are written exactly once by rank 0 before the first time step. All subsequent write operations are `append` calls that pre-allocate the trailing dimension, which avoids costly header rewrites. Dates are stored as

- **days since 1970-01-01 00:00:00** for surface stations,
- **milliseconds since 2019-01-01 11:14:35.629** for TROPOMI CH_4,

mirroring the conventions of the respective upstream data providers.

The station input file needs to follow the CF-1.8 NetCDF standard and to include the following variables:

- stime, etime: start/end of the observation interval (in days since epoch)
- latitude, longitude, elevation, sampling_height
- sampling_strategy: integer flag (see Section 2.3.6)
- site_name: station ID string

The header of an example station input file is shown in the Appendix. The station input file specifies not only location and time, but also the `sampling_height`, i.e. the height above the local surface where the observation was taken. This allows the plugin to interpolate model values vertically to the actual measurement altitude rather than assuming surface level.

2.4.6 Upper atmosphere extension with CAMS fields

ICON model simulations typically extend up to pressures which may not cover the full vertical range to which the satellite is sensitive.

To address this, the plugin supports vertical extension of ICON fields using global reanalysis data from CAMS. In our case, we used global CAMS CH_4 3D reanalysis fields up to very low pressures at fixed timestamps.

Every 6 hours, the plugin loads two CAMS snapshots that bracket the current model time and interpolates them linearly in time. The CAMS field is then used to fill in vertical layers above the ICON model top.

This vertical extension ensures that averaging kernels (which may span the entire atmospheric column) are applied over a physically meaningful vertical range. Without this extension, the topmost AK layers would fall outside the domain, biasing the resulting XCH_4 estimate.

The CAMS data is optional and only required when:

- tracking CH_4 satellite data, and
- applying averaging kernels whose sensitivity range extends above the ICON model top (this occurs when the simulation does not reach the top of the atmosphere, unless explicitly configured to do so)

2.4.7 Take-away

The chosen code design minimizes Python overhead, ensures good parallel scaling, and clearly separates ICON–ComIn interaction from the numerical kernels that perform the heavy computation.

2.5 Interpolation Methods

Interpolation is required whenever model fields need to be evaluated at observation locations that do not coincide with ICON's native grid. The plugin applies a combination of horizontal and vertical interpolation strategies, tailored to the type of observation.

2.5.1 Horizontal Interpolation

Two approaches are implemented, depending on the information available about the observation footprint:

2.5.1.1 Point-based interpolation (stations, satellite CIF) For monitoring stations, the geographic location is typically represented by a single point (latitude/longitude). In the case of satellite CIF data, only the footprint center coordinates (lat/lon) are provided by CIF, without full pixel geometry. In both cases, a fixed number of nearby ICON cell centres (default

$k = 4$) are identified using a `KDTree`. Their values are combined using inverse-distance weighting:

$$f(\vec{x}) = \sum_{i=1}^{k} w_i f_i \quad \text{with} \quad w_i = \frac{1/d_i}{\sum_j 1/d_j}$$

where d_i is the haversine distance from the target point $\vec{x}$ to the i-th cell center. This method is widely used in atmospheric science because it is computationally efficient and provides smooth transitions between neighbouring cells.

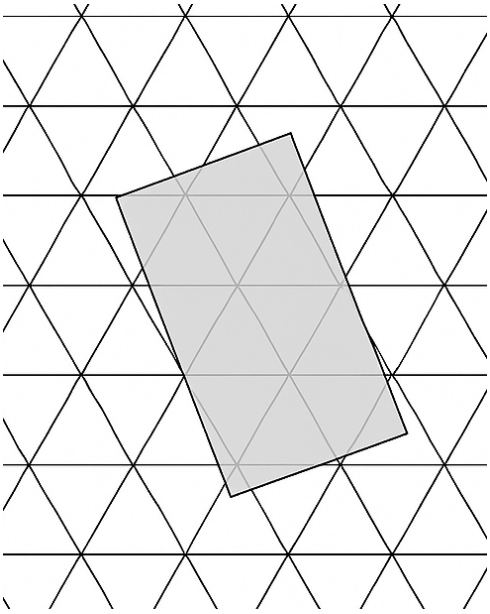

Figure 2: Figure showing the principle of a satellite pixel on an ICON grid.

2.5.1.2 Area-based interpolation (satellite CH_4) For satellite retrievals, often the full pixel geometry (corners in lat/lon) is available, see Figure 2. Here, a geometrical overlap is computed between the satellite footprint and the ICON cells. Each model cell contributes proportionally to its overlapping area with the footprint. However, one small caveat: As we divide our sampling work up into the different PE's, it could be that the actual satellite pixel goes over the boundary of the PE the sampling was assigned to. To monitor this, a variable has been added showing what percentage of the total area of the satellite pixel has been covered by our plugin. In CPU-based runs this value was observed to be occasionally slightly below 100 % when a footprint spans multiple processor boundaries. We expect this effect to be even smaller on GPU-based runs, where the domain decomposition uses far fewer processing elements.

2.5.2 Vertical Interpolation

All vertical interpolation is performed along model levels. For height-based coordinates (z), linear interpolation is applied directly between the two enclosing layers:

$$f(z) = f_{\text{low}} + \frac{z - z_{\text{low}}}{z_{\text{high}} - z_{\text{low}}} \left(f_{\text{high}} - f_{\text{low}}\right).$$

For pressure-based coordinates, interpolation is instead performed linearly in the logarithm of pressure to better represent the vertical structure of the atmosphere:

$$f(p) = f_{\text{low}} + \frac{\log(p) - \log(p_{\text{low}})}{\log(p_{\text{high}}) - \log(p_{\text{low}})} \left(f_{\text{high}} - f_{\text{low}}\right),$$

where p_{low} and p_{high} are the surrounding pressure levels.

This method is used for all observation types and is consistent with the vertical representation of ICON fields.

2.5.2.1 Points outside the lower or upper model boundaries

If the target height lies within the vertical model domain, values are linearly interpolated between the two nearest enclosing layers. However, if the target is outside this range (above the topmost or below the bottommost layer), the plugin does *not* extrapolate. Instead, it uses the value of the nearest valid layer:

- **Above model top**: take value from the highest model level.
- **Below surface**: take value from the lowest model level.

2.5.3 Application of Averaging Kernels

For satellite data such as observations of CH_4, the plugin can optionally apply column averaging kernels (AKs) to match model vertical profiles to the sensitivity of the satellite retrieval.

Each AK defines a weighting function over pressure levels, reflecting how much information the satellite provides at different pressures. After interpolating the model CH_4 profile $x_{\text{mod}}(p)$ onto the retrieval pressure grid, the AK is applied to the difference between the model profile and the a priori profile $x_{\text{a}}(p)$, according to the retrieval equation:

$$x_{\text{ret}} = x_{\text{a}} + \mathbf{A}\,(x_{\text{mod}} - x_{\text{a}})\,,$$

where $\mathbf{A}$ is the averaging kernel matrix, x_{a} is the a priori profile, and x_{ret} is the model-equivalent retrieved profile. This formulation ensures that the transformation is consistent with the retrieval's sensitivity structure, rather than a simple vertical weighting of concentrations. Finally, the retrieved profile x_{ret} is integrated over pressure using the retrieval's pressure-weight function to yield a single column-averaged value directly comparable to the satellite product.

The AKs and retrieval pressure grids are read from NetCDF files associated with each satellite footprint.

2.5.3.1 Pre-computation and reuse

To ensure performance and reproducibility, horizontal interpolation weights are pre-computed only once at startup and then used when needed. Vertical interpolation, however, depends on the target coordinate system:

- For height-based interpolation (e.g. station data), layer heights from `hhl` are available at startup, so vertical indices and weights can be pre-computed.

- For pressure-based interpolation (e.g. satellite data), vertical pressure profiles evolve over time and interpolation is carried out linearly in $\log(p)$. Therefore, interpolation indices and weights must be computed when they are sampled.

This hybrid strategy balances efficiency with physical accuracy, ensuring that each observation type is handled in a way consistent with its geophysical context.

2.6 Efficiency

The plugin is designed to minimize computational overhead and memory use, ensuring that observation tracking can be performed alongside ICON's own model integration without degrading performance. Several strategies are employed to achieve this.

2.6.1 MPI

All work is distributed across MPI ranks, following ICON's native parallel decomposition. Each observation (station or satellite pixel) is assigned to a single rank at startup, based on proximity to model cells. From that point onward, each rank processes only its own slice of the data, avoiding duplication or gaps. The only synchronisation occurs during batch output: once per write interval, each rank sends its completed data to rank 0 for NetCDF writing. This design ensures fully local memory access during computation, no redundant work across ranks, and good scalability across multiple nodes.

2.6.2 Precomputation

Where possible, lookup tables and interpolation weights are computed once at the start of the simulation and reused at each step. This applies in particular to horizontal weights for stations and point-based satellite CIF data, vertical layer indices for height-based interpolation, and arithmetic operator expressions defined in the YAML configuration. Only pressure-based vertical interpolation (e.g., CH_4 satellite retrievals) requires indices to be recomputed dynamically, since the vertical pressure grid evolves in time.

2.6.3 Vectorization

Most numerical operations in the plugin are implemented with NumPy vector routines, avoiding explicit Python loops and benefiting from compiled performance. This approach is used for horizontal and vertical interpolation, averaging, masking, and preparation of output arrays. For more complex tasks, such as computing geometric overlaps of CH_4 satellite footprints with ICON cells, short and memory-local loops are employed. Even in these cases, performance is maintained by operating on pre-filtered data slices.

Together, these strategies ensure that the plugin adds minimal runtime overhead, scales well with MPI parallelization, and maintains predictable memory usage.

2.7 Modularity

A key design principle of the plugin is modularity. By separating different observation types and processing steps into distinct components, the system remains flexible, maintainable, and easy to extend.

2.7.1 Independent satellite and station modules

Each observation type is implemented as an independent module with its own data container classes, interpolation strategy, tracking logic, and output format. This separation ensures that new observation types can be added without interfering with existing functionality. For example, the satellite CH_4 module can be disabled entirely without affecting monitoring station processing, and vice versa.

2.7.2 Support for any 3D ICON variable

The plugin can extract and process any three-dimensional variable available through the ComIn interface. This includes both ICON meteorological variables as well as ART tracers. Variables are specified at runtime in the YAML configuration file, including optional arithmetic expressions (e.g. `NO2 + NO`, `A - B`). Internally, a generic interface requests and reshapes the variables without hardcoding their names or meanings, ensuring flexibility across model setups.

2.7.3 Separation of configuration and code

All observational metadata (e.g., coordinates, pixel geometries, pressure grids, averaging kernels) is provided via external NetCDF files, while all user-defined choices (variables, formulas, writeout intervals, vertical strategies) are set in the YAML configuration file. This separation allows scientific setups to be modified without any changes to the source code, improving reproducibility and user accessibility.

2.7.4 Extensible output structure

Output files follow a flexible NetCDF structure, with dimensions and metadata that can accommodate new fields without breaking compatibility. Because the output layer is abstracted from the numerical processing, additional variables or quality-control flags can be included by updating the schema rather than altering the tracking logic.

Together, these modular design choices ensure that the plugin can support a wide range of use cases, from simple station monitoring to complex satellite emulation, while remaining robust against future extensions.

2.8 Interface

The plugin is fully controlled via a YAML configuration file, which defines all user choices regarding tracked variables, observation types, and output specifications. This design ensures a clear separation between scientific configuration and program logic.

2.8.1 YAML configuration file

The YAML file specifies:

- **Variables to track:** a list of ICON field names (e.g. `O3`, `temp`), optionally combined via expressions (e.g. `NO2 + NO`).
- **Observation types:** toggle inclusion of monitoring stations, satellite, and satellite CH_4.
- **Neighbour search and interpolation:**
  - `NUMBER_OF_NN`: number of nearest neighbours used for horizontal interpolation when no area information is available.
  - `accepted_distance` [km]: maximum allowed distance from the target point to the nearest cell in the grid; observations farther away are discarded. Should be set slightly larger than the horizontal grid spacing.
- **Decomposition / domain selection:**
  - `jg`: ICON domain (grid) index processed by the plugin.
  - `msgrank`: MPI rank responsible for I/O and orchestration within the plugin.
- **Writeout interval:** temporal resolution of output (in seconds), `time_interval_writeout`.
- **Input and output file paths:** paths to required NetCDF inputs (station locations, satellite footprint definitions, pressure grids, averaging kernels) and file names/paths for the plugin outputs (stations, satellite, satellite CH_4).

A template YAML file provided with the plugin serves as both documentation and a validation schema.

2.8.2 Runtime flexibility

The same codebase can be reused across experiments simply by changing the YAML file, without recompilation or code modification. This allows batch experiments with different variables or configurations, as well as selective enabling or disabling of observation types at runtime.

2.8.3 Minimal ComIn surface

On the ICON side, the plugin registers only a minimal set of ComIn callbacks and does not require additional Fortran interfaces. All user control is handled through the YAML configuration, avoiding dependence on ICON's namelist and reducing coupling with the model's input structures.

This interface design maximises flexibility, reproducibility, and ease of use while ensuring minimal integration effort on the ICON side.

2.9 Testcase

To demonstrate the functionality of the plugin, a testcase is provided through a public GitLab repository(see Section 5). This testcase allows users to reproduce the workflow described in this study and serves as a template for adapting the plugin to other regions, resolutions, or observational setups.

2.9.1 Contents

- A small ICON simulation over Romania with a horizontal resolution of 6 km,
- All required observation inputs (monitoring station locations, satellite CIF and CH_4 data, averaging kernels, etc.),
- A pre-configured YAML file for tracking a minimal set of variables,
- A `notes.txt` file with instructions on how to compile, configure, and run the plugin with this data.

2.9.2 Getting started

Users can download the testcase directly from the GitLab repository(see Section 5).

The accompanying instructions describe how to compile ICON with the plugin enabled, adjust paths in the YAML file, run a short simulation, and inspect the resulting NetCDF output files.

2.9.3 Purpose

The testcase is designed as a functional testbed rather than a full scientific experiment. It allows new users to verify that:

- the plugin compiles and runs correctly in their environment,
- the interpolation and tracking routines operate as intended, and

- the YAML interface and NetCDF outputs are understood.

By providing this testcase, the plugin ensures immediate reproducibility and lowers the entry barrier for adoption in future scientific or operational studies.

3 Results and Discussion

To demonstrate the capabilities and flexibility of the tracking plugin, we present a series of test cases using synthetic or realistic ICON simulations. These examples show how the plugin can be used to extract observationally relevant data from model runs, with minimal setup and configuration.

Since the focus of this work is infrastructure and method development, we limit the discussion to structural validation and performance, rather than detailed scientific interpretation. Each subsection illustrates one of the supported data types introduced in Section 2.3.

3.1 Monitoring station data

As a first example, we track the near surface level concentration of CH_4 and temperature at fixed geographic locations, emulating data collected by air quality monitoring stations. The plugin is configured to average model fields over 1-hour intervals, which is typical for greenhouse gas concentration records from, for example, ICOS stations.

It should be noted that station measurements can occur at arbitrary heights above ground (or sea) (e.g. rooftop or tower stations). The plugin accounts for this by interpolating vertically to the reported sampling height of each station.

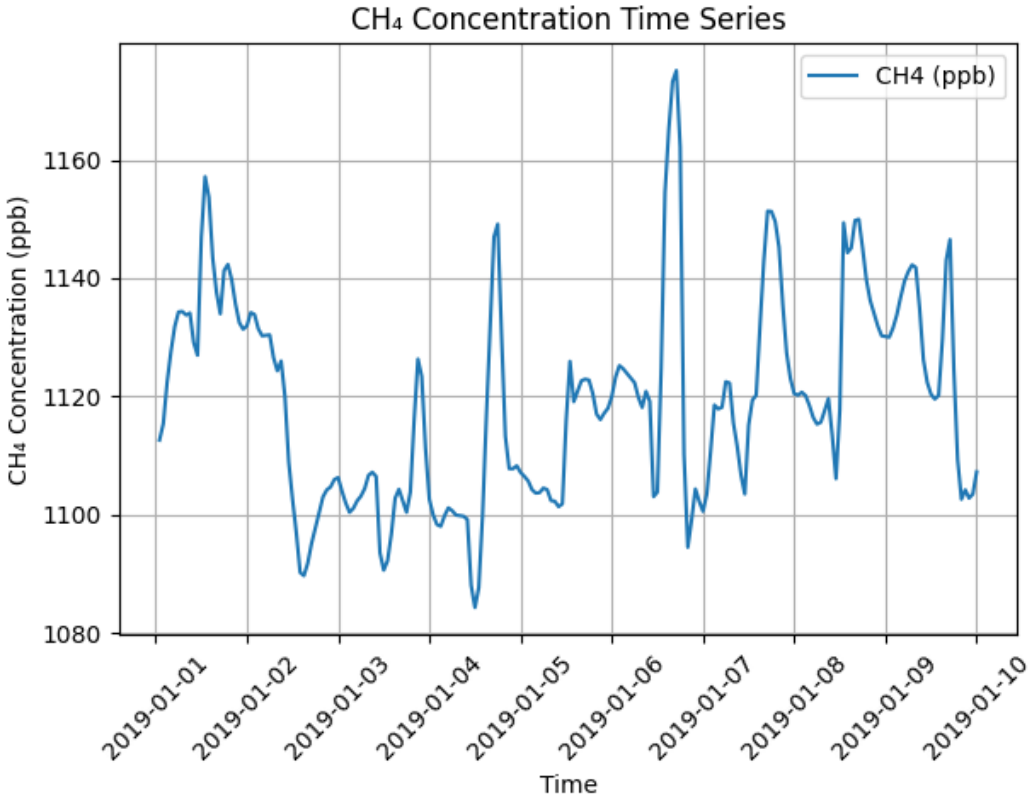

Figure 3: Time series of simulated CH_4 mixing ratios at a fictive monitoring station in Romania, extracted with one-hour averaging. The plugin produces continuous output aligned with the defined station configuration.

Figure 3 shows an example time series of CH4 extracted at one location in the Romanian domain.

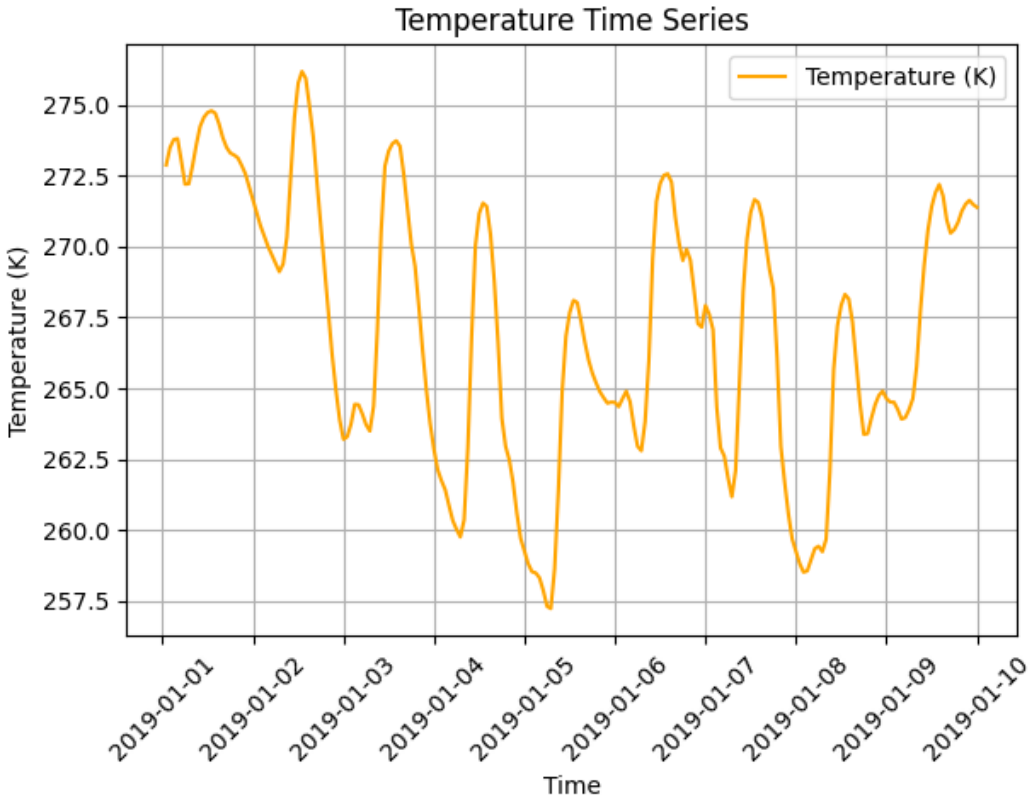

Figure 4: Time series of simulated near-surface temperature at the same station, averaged over one-hour intervals. The consistent coverage demonstrates reliable performance for multiple variables.

Figure 4 shows an example time series of temperature extracted at one location in the Romanian domain.

This example demonstrates the ability of the plugin to:

- sample arbitrary 3D model fields at ground level,
- average over user-defined time intervals,
- output structured NetCDF files per station.

3.2 Mobile sampling: aircraft-style flight

To evaluate the plugin's performance against real-world data, we applied it to a measured aircraft trajectory over Romania. The input consisted of recorded 3D coordinates and timestamps from the flight, along with in situ CH_4 observations. The plugin sampled ICON output along the same trajectory at the model's dynamical timestep (Δt_{model}), ensuring maximal temporal fidelity, without averaging, to enable a direct point-by-point comparison with the measurements.

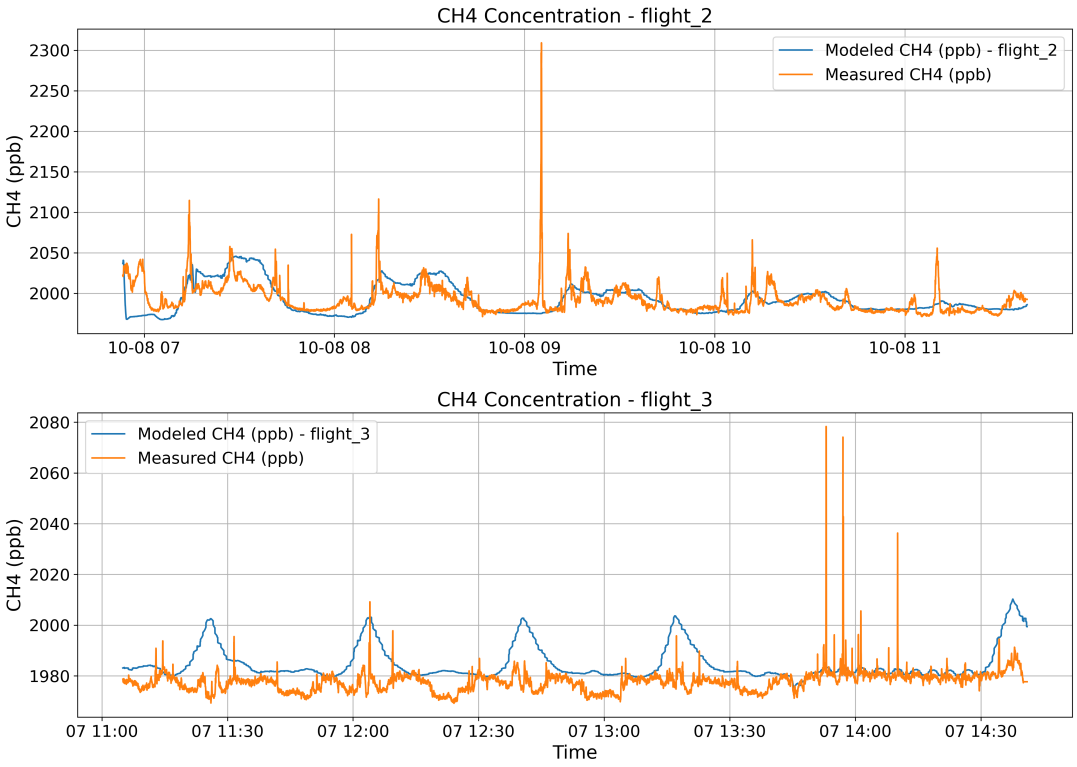

Figure 5: Comparison of observed CH_4 mixing ratios from an aircraft flight over Romania (black) with ICON model output extracted along the same trajectory using the plugin (red). The results demonstrate the ability to generate observation-equivalent time series for direct validation against in situ measurements.

Figure 5 compares the observed CH_4 mixing ratios with the corresponding modelled values extracted by the plugin. The simulated profile reproduces the overall temporal and spatial variability along the flight path, with deviations primarily linked to known model issues such as inaccuracies in the depth of the atmospheric boundary layer rather than sampling errors. Crucially, the comparison with real data is equivalent —and in fact more temporally accurate —than the traditional approach (see Section 3.4), since the plugin samples directly at observation times without interpolation.

This comparison highlights the plugin's ability to:

- accurately reproduce measured flight paths using ICON fields,
- produce observation-equivalent datasets directly comparable with in situ aircraft data, and
- enable validation of ICON simulations in realistic campaign scenarios with temporal resolutions that would not be possible with the traditional approach.

3.3 Satellite-style data: TROPOMI CH_4

To test the plugin in a satellite context, we applied it to column-averaged CH_4 retrievals from TROPOMI for January 2019. Each satellite footprint provided the necessary corner geometry, retrieval pressure grid, and averaging kernel.

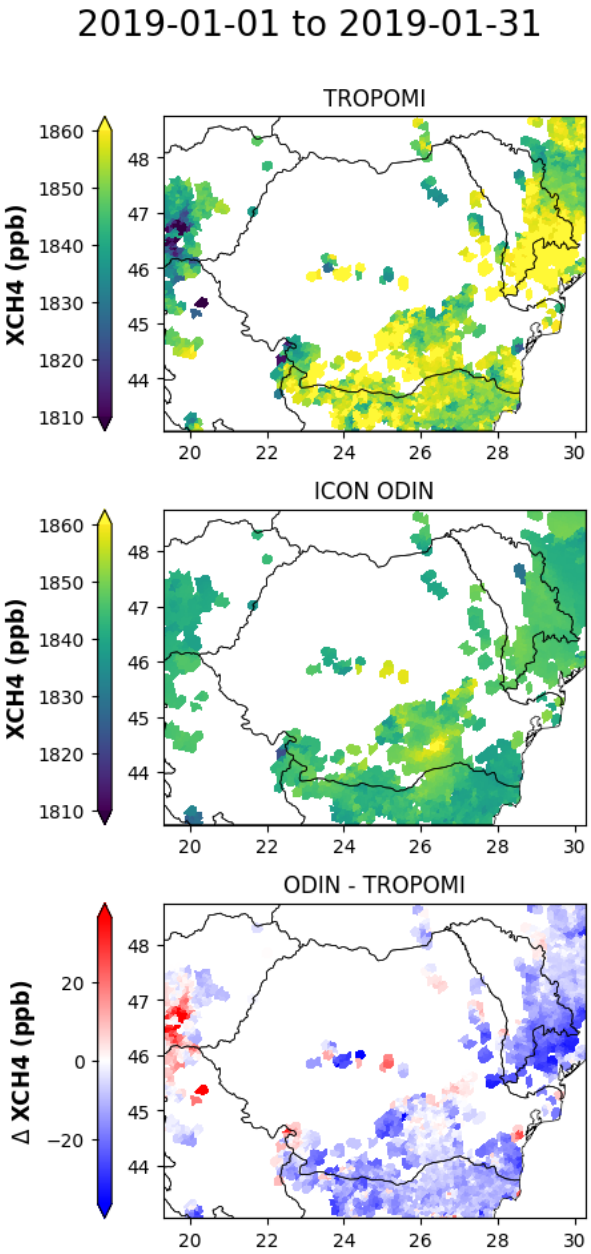

Figure 6: Comparison of column-averaged CH_4 (XCH_4) over Romania in January 2019: TROPOMI retrievals (top), ICON model output processed with the plugin (middle) and the difference (bottom). The plugin applies vertical interpolation, CAMS-based profile extension, and averaging kernels to produce observation-equivalent satellite data.

The plugin constructed full vertical CH_4 profiles by combining ICON outputs with CAMS reanalysis data to extend the model above the ICON top. The profiles were interpolated onto the retrieval pressure grids, and the averaging kernels were applied within the plugin to generate observation-equivalent XCH_4 values.

Figure 6 compares the plugin-derived XCH_4 with the corresponding TROPOMI retrievals. The results demonstrate that the plugin can reproduce satellite-style column values directly during runtime, with temporal alignment to the measurement and without offline interpolation, thereby avoiding postprocessing altogether. Spatial differences reflect underlying model–observation discrepancies rather than methodological limitations.

This case study confirms the plugin's ability to:

- handle complete satellite processing workflows (geometry, profile extension, averaging kernel application),
- generate column values directly comparable with TROPOMI retrievals, and
- integrate CAMS data seamlessly for vertical profile extension beyond the ICON model top.

3.4 Comparison with traditional postprocessing

The standard approach to obtaining satellite-equivalent values from ICON simulations is offline postprocessing: the model outputs are written at hourly intervals, linearly interpolated in time, and subsequently processed with satellite retrieval data. To assess the benefit of direct in-plugin sampling, we compared results from this traditional workflow with those obtained using the plugin.

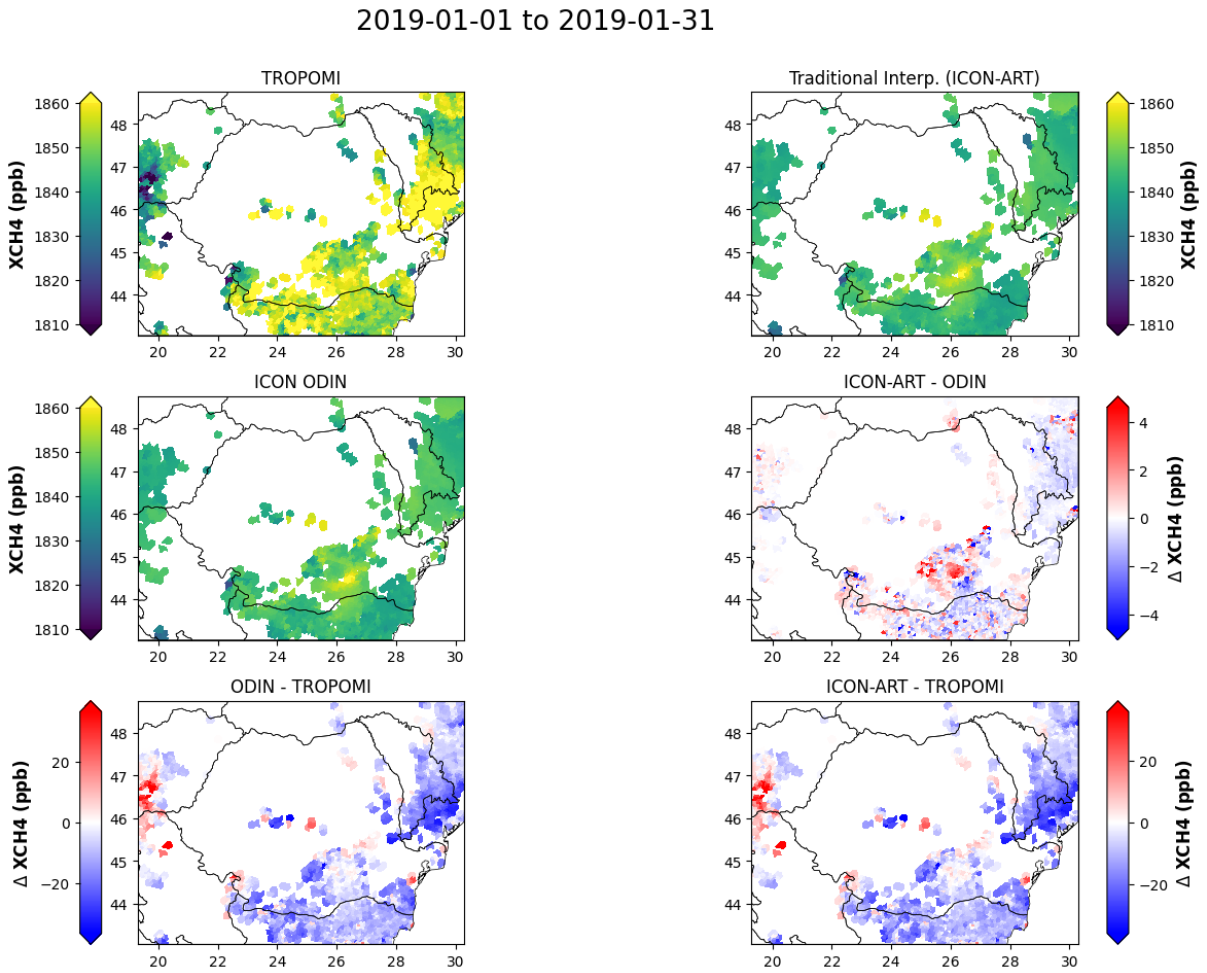

Figure 7: Differences in column-averaged CH_4 (XCH_4) between the plugin-based sampling and the traditional offline interpolation workflow, with TROPOMI retrievals shown for reference. The plugin avoids errors introduced by temporal interpolation between model outputs.

Figure 7 shows the spatial distribution of differences between the two approaches, alongside TROPOMI retrievals for reference. The plugin-derived values exhibit systematic differences of up to 4 ppb compared with the offline method. These differences are primarily due to the plugin's ability to sample the model at the exact observation time, avoiding uncertainties introduced by temporal interpolation between output steps.

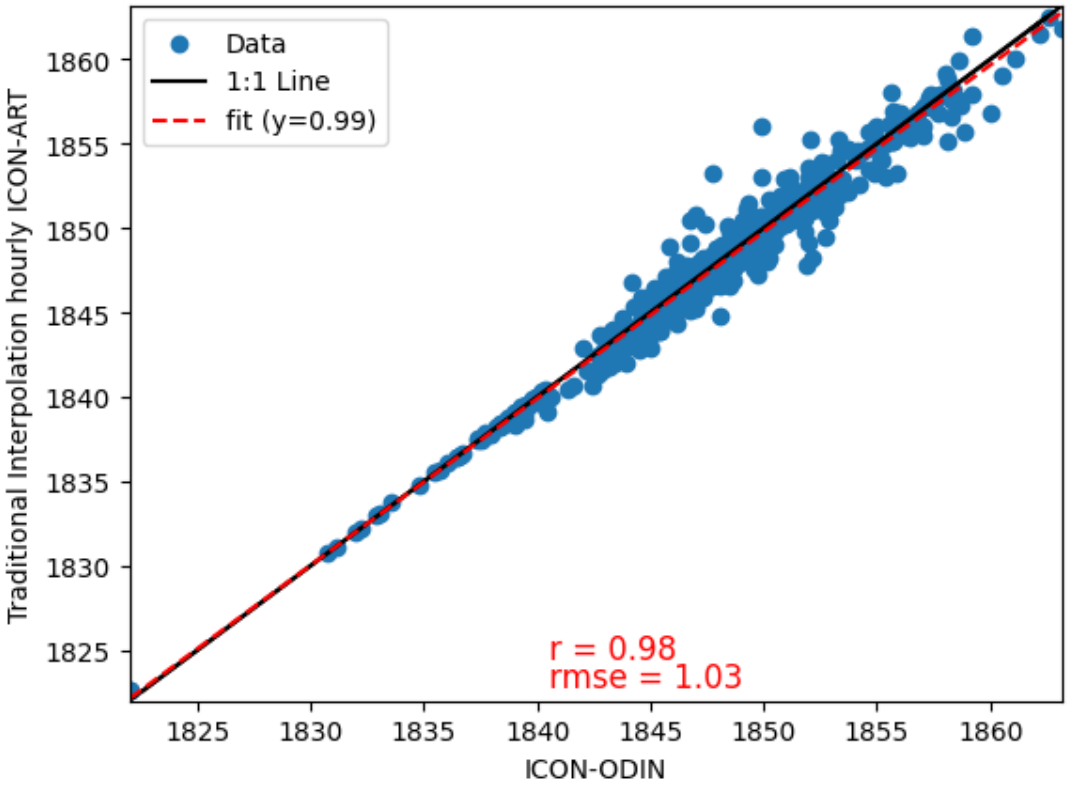

Figure 8: Scatterplot comparing plugin-derived and traditionally processed XCH_4 values. While no systematic bias is observed, the spread indicates improved temporal fidelity from in-plugin sampling (RMSE of around 1 ppb).

Figure 8 provides a scatterplot of plugin-derived versus traditionally processed values. The absence of systematic bias confirms consistency between the two methods, while the observed spread (root mean squared difference of around 1 ppb) indicates improved temporal fidelity of the in-plugin approach.

This comparison demonstrates that in-plugin sampling:

- yields results consistent with traditional methods,
- improves temporal accuracy by eliminating interpolation errors, and
- enhances reproducibility by removing the need for external postprocessing pipelines.

The differences of up to $\sim 4\,\text{ppb}$ in XCH_4 are substantial given that they arise solely from temporal interpolation in the traditional method—highlighting that accurate temporal sampling as provided by ODIN can have a significant impact.

3.5 Performance benchmarking

Beyond scientific accuracy, an important criterion for the plugin is its computational performance. Because the sampling is performed during model runtime, it is essential to ensure that the additional workload does not introduce significant overhead.

To evaluate performance, we compared ICON simulations with and without the plugin enabled on the Alps Eiger machine. In the plugin runs, all model output was disabled (no native-grid or remapped streams). In the control runs without the plugin, an hourly native-grid output stream was active and, in addition, a structured (remapped) stream wrote every two hours. Because the remapped stream can incur overhead from horizontal weight computation, we repeated two tests with this 2-hour stream disabled and observed no measurable change in runtime. We therefore retain the original configuration for the results reported below. Apart from these I/O settings, the configurations were identical; in the plugin runs, satellite tracking was enabled as the most demanding task.

The runtime measurements for five simulations each, with and without the plugin enabled, are presented in Table I and in Figure 9. The mean runtime without the plugin was 1570.6 ± 5.6 s (≈26 min 11 s), while with the plugin enabled it was 1581.4 ± 7.4 s (≈26 min 21 s). This corresponds to a difference of 10.8 s (0.69%). This difference is too small to conclude that there is a significant increase in runtime imposed by the plugin.

When using the plugin, there is no extra postprocessing step needed. The cost of this postprocessing step will vary substantially between use cases. When testing on our setup, this step took around two to three minutes, i.e. 120 to 180 seconds. In essence, we conclude that our plugin does not alter the performance or runtime in a measurable way but instead, by eliminating the need for postprocessing, it is computationally faster while also saving the space of the outputted files of ICON.

Table I: Runtime of ICON simulations with and without the plugin enabled. Times are reported in seconds.

Run	1	2	3	4	5	Mean ± SD
Without plugin	1572	1566	1573	1564	1578	1570.6 ± 5.6
With plugin	1574	1584	1573	1588	1588	1581.4 ± 7.4

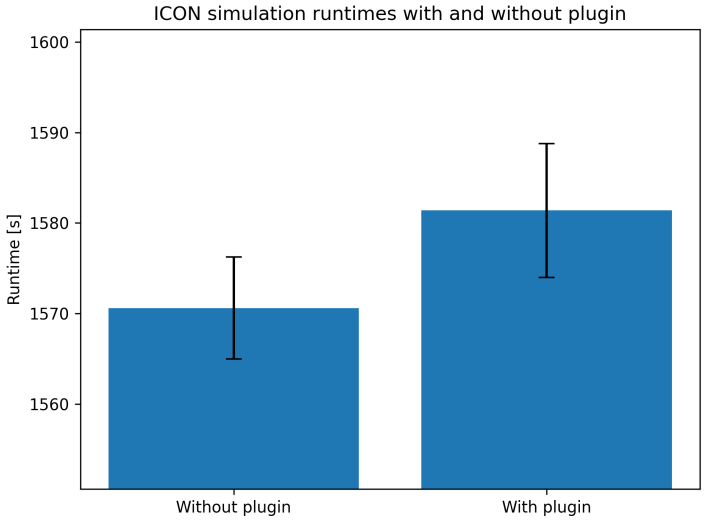

Figure 9: Mean runtimes of ICON simulations with and without the plugin enabled. Error bars indicate the standard deviation (Note: y-axis truncated to highlight small differences).

To evaluate the difference in runtime in more detail, we present in the appendix the time

each module took in ICON when running it with both the outputting of files and with the plugin turned on.

4 Conclusions

In this work, a new plugin for the ICON model was developed and demonstrated, enabling in-simulation sampling of atmospheric variables through the ComIn interface. By directly extracting variables at specified spatio-temporal locations during runtime, the plugin eliminates the need to store complete model fields and avoids interpolation errors introduced by traditional offline postprocessing.

The plugin was successfully applied to multiple observation types:

- Monitoring stations, such as ICOS sites, where it reproduced observation-style time series with user-defined averaging,
- Aircraft trajectories, where it generated observation-equivalent datasets directly comparable with in situ measurements, and
- Satellite retrievals, where it handled vertical interpolation, stratospheric extension with CAMS data, and averaging-kernel application to produce TROPOMI-equivalent CH_4 columns.

The plugin is designed for ease of use and extensibility: users define their tracking targets and configuration through a YAML file, without the need to modify the codebase. This makes the plugin suitable for a wide range of use cases, including model evaluation, data assimilation, and inversion frameworks such as CIF.

Preliminary results show that the performance of ICON is not affected by our plugin when turning off the output of ICON.

A few limitations of the current implementation should be noted:

- The plugin currently supports only **domain 1** of ICON. Nested or multi-domain setups are not yet implemented.
- All sampling points, locations, times, and variables, must be defined before the simulation starts. If the user wishes to change the sampling setup, the model must be rerun.
- For satellite-type sampling, users can also perform postprocessing within the plugin, including stratospheric profile extension (as demonstrated here with CAMS and CH_4). Implementing other postprocessing setups currently requires adapting the source code.

Despite these constraints, the plugin provides a powerful framework for in-simulation data extraction. It avoids the need for external postprocessing and, for satellite datasets, improves temporal accuracy.

Future work may include support for nested domains, generalization of the in-plugin satellite postprocessing to variables beyond CH_4, runtime-configurable profile extensions without code modification, full vectorization of all computations, and enabling processor information exchange so that entire satellite footprints can be processed even when crossing processor boundaries. In addition, alternative or more advanced interpolation methods could be implemented, potentially also operating across processor boundaries.

In summary, this plugin provides a robust and generalisable foundation for extracting observation-equivalent data directly from ICON simulations, with minimal effort, flexible configuration, and strong performance.

5 Data and Code Availability

The ICON plugin developed in this work, including its source code, configuration files, and a testcase demonstrating its functionality, is available at: https://gitlab.empa.ch/abt503/users/huze/comin-python.git. The lateral boundary conditions are not available there, they are stored on the Alps Eiger machine. Please contact Dominik Brunner via dominik.brunner@empa.ch for access to those files.

The testcase reproduces the sampling configurations described in Section 2.9 and includes example YAML configuration files, input data for all supported observation types, and postprocessing scripts used to generate the figures in this thesis. The repository also contains benchmarking scripts and analysis notebooks. The paths to files will need to be changed when users want to use this testcase.

A Appendix

Header of an input file for the stationary measurements:

```
netcdf station_input {
dimensions:
        obs = UNLIMITED ; // (216 currently)
        nchar = 20 ;
variables:
        double stime(obs) ;
                stime:units = "days since 1970-01-01 00:00:00" ;
                stime:long_name = "start time of observation interval; UTC" ;
                stime:calendar = "proleptic_gregorian" ;
        double etime(obs) ;
                etime:units = "days since 1970-01-01 00:00:00" ;
                etime:long_name = "end time of observation interval; UTC" ;
                etime:calendar = "proleptic_gregorian" ;
        float longitude(obs) ;
                longitude:units = "degrees_east" ;
                longitude:standard_name = "longitude" ;
        float latitude(obs) ;
                latitude:units = "degrees_north" ;
                latitude:standard_name = "latitude" ;
        float elevation(obs) ;
                elevation:units = "m" ;
                elevation:long_name = "surface elevation above sea level" ;
        float sampling_height(obs) ;
                sampling_height:units = "m" ;
                sampling_height:long_name = "sampling height above surface" ;
        float sampling_strategy(obs) ;
                sampling_strategy:units = "1" ;
                sampling_strategy:long_name = "sampling strategy flag" ;
                sampling_strategy:comment = "1=low ; 2=mountain ; 3=instantaneous low; 4
        char site_name(obs, nchar) ;
                site_name:long_name = "station name or ID" ;

// global attributes:
                :Conventions = "CF-1.8" ;
                :title = "Station input file for ICON ComIn interface ODIN" ;
                :institution = "Empa" ;
                :source = "ICON ComIn interface ODIN" ;
                :version = "1.0" ;
                :author = "Zeno Hug" ;
                :transport_model = "ICON" ;
                :transport_model_version = "" ;
                :experiment = "" ;
                :project = "" ;
                :references = "" ;
                :comment = "" ;
                :license = "CC-BY-4.0" ;
                :history = "" ;
}
```

Timer report, ranks 123-125

name	# calls	t_min	min rank	t_avg	t_max	max rank	total min (s)	total min rank	total max (s)	total max rank	total avg (s)	# PEs
wrt_output	293	0.00000s	[0]	1.0021s	14.0042s	[0]	0.000	[2]	262.893	[0]	97.876	3

Timer report, ranks 0-122

name	# calls	t_min	min rank	t_avg	t_max	max rank	total min (s)	total min rank	total max (s)	total max rank	total avg (s)	# PEs
model_init	3	0.00010s	[1]	0.00029s	0.00065s	[0]	0.000	[1]	0.001	[0]	0.000	3
L compute_domain_decomp	3	0.00000s	[0]	0.00000s	0.00000s	[0]	0.000	[0]	0.000	[0]	0.000	3

name	# calls	t_min	min rank	t_avg	t_max	max rank	total min (s)	total min rank	total max (s)	total max rank	total avg (s)	# PEs
total	123	28m16s	[52]	28m16s	28m16s	[122]	1696.739	[52]	1696.741	[122]	1696.740	123
L integrate_nh	1416960	0.11447s	[33]	0.13299s	1.7871s	[45]	1523.298	[81]	1536.151	[28]	1532.026	123
L nh_solve	7084800	0.00471s	[0]	0.01372s	1.5879s	[27]	662.222	[0]	830.220	[83]	790.262	123
L nh_solve.veltend	8501760	0.00027s	[122]	0.00105s	0.00616s	[4]	28.185	[122]	94.556	[24]	72.400	123
L nh_solve.cellcomp	14169600	0.00015s	[121]	0.00072s	0.00661s	[4]	26.349	[121]	118.772	[56]	83.008	123
L nh_solve.edgecomp	14169600	0.00012s	[121]	0.00105s	0.00724s	[4]	33.027	[121]	168.384	[22]	121.062	123
L nh_solve.vnupd	14169600	0.00010s	[81]	0.00096s	0.00753s	[4]	39.319	[121]	143.199	[10]	110.799	123
L nh_solve.vimpl	14169600	0.00030s	[122]	0.00108s	0.00677s	[4]	44.072	[122]	164.380	[16]	123.917	123
L nh_solve.exch	28339200	0.00004s	[0]	0.00118s	1.5791s	[65]	87.706	[16]	578.503	[121]	270.764	123
L nh_hdiff	1417083	0.00129s	[0]	0.00290s	0.05338s	[0]	24.183	[83]	80.756	[120]	33.459	123
L transport	1416960	0.01861s	[31]	0.02542s	0.03749s	[122]	283.307	[121]	301.200	[2]	292.841	123
L adv_horiz	1416960	0.01371s	[101]	0.01919s	0.02728s	[97]	200.071	[0]	234.641	[33]	221.115	123
L adv_hflx	1416960	0.01228s	[101]	0.01772s	0.02567s	[1]	192.155	[0]	226.242	[122]	204.190	123
L back_traj	4250880	0.00000s	[0]	0.00041s	0.00495s	[4]	9.152	[115]	16.354	[58]	14.017	123
L adv_vert	1416960	0.00183s	[81]	0.00533s	0.01414s	[101]	41.662	[122]	73.968	[0]	61.427	123
L adv_vflx	1416960	0.00169s	[81]	0.00417s	0.01321s	[101]	38.371	[122]	55.310	[0]	48.035	123
L action	1416960	0.00001s	[120]	0.00003s	0.00280s	[4]	0.200	[122]	0.516	[37]	0.339	123
exch_data	58094499	0.00000s	[6]	0.00097s	1.5791s	[65]	207.670	[16]	966.893	[122]	460.106	123
L exch_data.wait	58094499	0.00000s	[0]	0.00084s	1.5790s	[65]	134.365	[16]	940.211	[122]	398.028	123
wrt_output	23739	0.00258s	[52]	0.16563s	6.0781s	[122]	31.345	[52]	32.780	[122]	31.967	123
L wait_for_async_io	23739	0.00001s	[48]	0.16217s	6.0738s	[122]	30.676	[52]	32.112	[122]	31.299	123
physics	1417083	0.01217s	[81]	0.02864s	0.14007s	[55]	307.156	[81]	342.082	[30]	329.989	123
L nwp_radiation	141819	0.02109s	[102]	0.04681s	0.11596s	[103]	47.955	[102]	65.607	[81]	53.967	123
L preradiaton	141819	0.00121s	[59]	0.00151s	0.00634s	[36]	1.572	[117]	2.231	[0]	1.740	123
L radiation	408162	0.00096s	[91]	0.01308s	0.04028s	[44]	26.382	[122]	54.002	[80]	43.420	123
L rrtm_prep	408162	0.00011s	[112]	0.00074s	0.02266s	[122]	1.291	[121]	3.079	[94]	2.461	123
L rrtm_post	408162	0.00000s	[112]	0.00001s	0.00035s	[1]	0.013	[121]	0.040	[94]	0.031	123
L lrtm	408162	0.00073s	[112]	0.00669s	0.01973s	[8]	12.879	[122]	28.310	[80]	22.202	123
L srtm	408162	0.00000s	[5]	0.00549s	0.02769s	[58]	11.639	[42]	23.064	[64]	18.212	123
L phys_acc_sync	1417083	0.00026s	[122]	0.00638s	0.07344s	[104]	33.494	[6]	136.064	[122]	73.544	123
L satad	1417083	0.00011s	[108]	0.00167s	0.00558s	[4]	15.415	[121]	22.069	[82]	19.195	123
L phys_u_v	1417083	0.00010s	[58]	0.00026s	0.00182s	[4]	1.770	[121]	3.506	[42]	2.952	123
L nwp_turbulence	2834043	0.00010s	[81]	0.00229s	0.02692s	[52]	32.058	[81]	62.144	[6]	52.809	123
L nwp_turbtrans	1417083	0.00009s	[81]	0.00052s	0.00487s	[4]	1.266	[122]	9.305	[98]	5.969	123
L nwp_turbdiff	13132800	0.00005s	[80]	0.00026s	0.00436s	[4]	17.462	[81]	31.304	[27]	27.589	123
L nwp_surface	1416960	0.00007s	[81]	0.00143s	0.01451s	[5]	1.120	[81]	30.467	[43]	16.506	123
L nwp_microphysics	1416960	0.00104s	[0]	0.00348s	0.01038s	[111]	20.776	[0]	55.425	[58]	40.037	123
L rediag_prog_vars	1417083	0.00011s	[0]	0.00068s	0.00779s	[4]	4.032	[122]	10.107	[6]	7.836	123
L nwp_convection	708603	0.00268s	[0]	0.00669s	0.01510s	[10]	21.638	[0]	51.087	[58]	38.536	123
L sso	236283	0.00021s	[81]	0.00082s	0.00395s	[84]	0.421	[81]	2.206	[25]	1.579	123
L cloud_cover	708603	0.00125s	[81]	0.00158s	0.00481s	[97]	8.837	[121]	9.603	[60]	9.125	123
L radheat	1417083	0.00023s	[81]	0.00055s	0.00517s	[4]	3.159	[122]	8.350	[14]	6.387	123
nh_diagnostics	5741394	0.00000s	[0]	0.00004s	0.00547s	[70]	1.041	[122]	2.250	[12]	1.792	123
diagnose_pres_temp	733326	0.00000s	[0]	0.00004s	0.00218s	[4]	0.160	[121]	0.386	[0]	0.261	123
model_init	369	0.41431s	[121]	18.9399s	44.9148s	[15]	56.810	[26]	56.847	[70]	56.820	123
L global_sum	984	0.00000s	[0]	0.00106s	0.03763s	[70]	0.000	[37]	0.040	[70]	0.008	123

L compute_domain_decomp	123	0.69956s	[10]	0.70219s	0.70574s	[15]	0.700	[10]	0.706	[15]	0.702	123
L ordglb_sum	2583	0.00001s	[16]	0.00013s	0.00342s	[0]	0.002	[116]	0.004	[0]	0.003	123
L compute_intp_coeffs	123	0.27673s	[118]	0.27941s	0.28147s	[62]	0.277	[118]	0.281	[62]	0.279	123
L lonlat_setup	123	0.01960s	[7]	0.02038s	0.02148s	[8]	0.020	[7]	0.021	[8]	0.020	123
L init_ext_data	123	0.19180s	[2]	0.19397s	0.19716s	[9]	0.192	[2]	0.197	[9]	0.194	123
L init_icon	123	1.3580s	[107]	1.3614s	1.3645s	[114]	1.358	[107]	1.364	[114]	1.361	123
L init_latbc	123	1.7026s	[39]	1.7047s	1.7081s	[11]	1.703	[39]	1.708	[11]	1.705	123
L init_nwp_phy	123	0.29465s	[51]	0.31028s	0.32241s	[0]	0.295	[51]	0.322	[0]	0.310	123
L art_initInt	369	0.00000s	[0]	0.15137s	0.50907s	[16]	0.231	[72]	0.510	[37]	0.454	123
L art_tracInt	492	0.00141s	[12]	0.00314s	0.00695s	[1]	0.010	[117]	0.016	[1]	0.013	123
L comin_init	861	0.00000s	[0]	0.04213s	0.31402s	[48]	0.268	[121]	0.314	[48]	0.295	123
L comin_primary_constructors	123	43.6603s	[66]	43.6634s	43.6677s	[113]	43.660	[66]	43.668	[113]	43.663	123
upper_atmosphere	492	0.00000s	[53]	0.00020s	0.00135s	[113]	0.000	[0]	0.002	[113]	0.001	123
L upatmo_construction	246	0.00001s	[54]	0.00035s	0.00135s	[113]	0.000	[21]	0.002	[113]	0.001	123
L upatmo_destruction	246	0.00000s	[51]	0.00005s	0.00035s	[64]	0.000	[0]	0.000	[64]	0.000	123
write_restart	1230	0.01158s	[0]	0.17043s	1.5649s	[6]	1.697	[0]	1.715	[121]	1.704	123
L write_restart_io	123	0.00000s	[0]	0.00000s	0.00000s	[44]	0.000	[0]	0.000	[44]	0.000	123
L write_restart_communication	2214	0.00000s	[0]	0.08727s	1.5643s	[66]	1.565	[0]	1.582	[121]	1.571	123
ART	34814838	0.00000s	[0]	0.00009s	0.07805s	[42]	22.392	[117]	38.021	[122]	26.312	123
L art_aeroInt	1416960	0.00000s	[2]	0.00000s	0.00025s	[38]	0.016	[2]	0.034	[65]	0.022	123
L art_emissInt	1416960	0.00044s	[4]	0.00168s	0.07805s	[42]	15.572	[117]	31.167	[122]	19.309	123
L art_reacInt	1416960	0.00039s	[92]	0.00048s	0.00226s	[33]	4.951	[33]	6.329	[0]	5.516	123
L art_diagInt	1440822	0.00000s	[0]	0.00000s	0.00211s	[23]	0.035	[104]	0.060	[80]	0.046	123
L art_sedInt	1416960	0.00000s	[0]	0.00000s	0.00011s	[54]	0.001	[50]	0.007	[122]	0.005	123
L art_toolInt	23616	0.00000s	[121]	0.00000s	0.00006s	[80]	0.000	[120]	0.001	[122]	0.000	123
L art_turbdiffInt	26265600	0.00000s	[19]	0.00001s	0.00082s	[18]	1.013	[42]	1.564	[88]	1.365	123
L art_washoutInt	1416960	0.00000s	[0]	0.00000s	0.00019s	[42]	0.001	[3]	0.004	[42]	0.002	123
comin_callbacks	39679431	0.00000s	[12]	0.00014s	7.6531s	[72]	43.400	[116]	49.600	[0]	45.382	123
optional_diagnostics_atmosphere	23739	0.00000s	[59]	0.00001s	0.00078s	[119]	0.001	[122]	0.003	[36]	0.002	123

B Statement on the Use of AI Tools

AI-based tools (ChatGPT by OpenAI) were used for grammar refinement, rephrasing of certain sentences, improving text clarity, and providing coding support (e.g., suggestions for plotting scripts). The scientific content, methodology, data analysis, and conclusions were solely developed by the author.

Bibliography

Antoine Berchet, Espen Sollum, Rona L. Thompson, Isabelle Pison, Joël Thanwerdas, Grégoire Broquet, Frédéric Chevallier, Tuula Aalto, Adrien Berchet, Peter Bergamaschi, Dominik Brunner, Richard Engelen, Audrey Fortems-Cheiney, Christoph Gerbig, Christine D. Groot Zwaaftink, Jean-Matthieu Haussaire, Stephan Henne, Sander Houweling, Ute Karstens, Werner L. Kutsch, Ingrid T. Luijkx, Guillaume Monteil, Paul I. Palmer, Jacob C. A. Van Peet, Wouter Peters, Philippe Peylin, Elise Potier, Christian Rödenbeck, Marielle Saunois, Marko Scholze, Aki Tsuruta, and Yuanhong Zhao. The community inversion framework v1.0: a unified system for atmospheric inversion studies. *Geoscientific Model Development*, 14(8):5331–5354, 2021. doi:10.5194/gmd-14-5331-2021. URL https://doi.org/10.5194/gmd-14-5331-2021.

Kerstin Hartung, Bastian Kern, Nils-Arne Dreier, Jörn Geisbüsch, Mahnoosh Haghighatnasab, Patrick Jöckel, Astrid Kerkweg, Wilton Jaciel Loch, Florian Prill, and Daniel Rieger. Icon comin – the icon community interface (comin version 0.1.0, with icon version 2024.01-01). *Geoscientific Model Development*, 18:1001–1015, 2025. doi:10.5194/gmd-18-1001-2025. URL https://doi.org/10.5194/gmd-18-1001-2025. © Author(s) 2025. Distributed under the Creative Commons Attribution 4.0 License.

J. Heiskanen, C. Brümmer, N. Buchmann, C. Calfapietra, H. Chen, B. Gielen, T. Gkritzalis, S. Hammer, S. Hartman, M. Herbst, I. A. Janssens, A. Jordan, E. Juurola, U. Karstens, V. Kasurinen, B. Kruijt, H. Lankreijer, I. Levin, M.-L. Linderson, D. Loustau, L. Merbold, K. Lund Myhre, D. Papale, M. Pavelka, K. Pilegaard, M. Ramonet, C. Rebmann, J. Rinne, L. Rivier, E. Saltikoff, R. Sanders, M. Steinbacher, T. Steinhoff, A. Watson, A. T. Vermeulen, T. Vesala, G. Vítková, and W. Kutsch. The integrated carbon observation system in europe. *Bulletin of the American Meteorological Society*, 103(3):E3–E13, 2022. doi:10.1175/BAMS-D-19-0364.1. URL https://doi.org/10.1175/BAMS-D-19-0364.1.

Anna Lönnqvist, Erik Sahlée, and Anna Rutgersson. Interpolating wind speed observations using a mesoscale model and meteorological tower measurements. *Journal of Atmospheric and Oceanic Technology*, 26(4):656–664, 2009. doi:10.1175/2008JTECHO588.1.

D. Rieger, M. Bangert, I. Bischoff-Gauss, J. Förstner, K. Lundgren, D. Reinert, J. Schröter, H. Vogel, G. Zängl, R. Ruhnke, and B. Vogel. Icon–art 1.0 – a new online-coupled model system from the global to regional scale. *Geoscientific Model Development*, 8(6):1659–1676, 2015. doi:10.5194/gmd-8-1659-2015. URL https://gmd.copernicus.org/articles/8/1659/2015/.

Rijaf, Communication, and Climate Data Tools Team. Interpolation methods in cdt: Inverse distance weighting. https://iri.columbia.edu/~rijaf/CDTUserGuide/html/interpolation_methods.html, 2020. Accessed: 2025-07-07.

J. Schröter, D. Rieger, C. Stassen, H. Vogel, M. Weimer, S. Werchner, J. Förstner, F. Prill, D. Reinert, G. Zängl, M. Giorgetta, R. Ruhnke, B. Vogel, and P. Braesicke. Icon-art 2.1: a flexible tracer framework and its application for composition studies in numerical weather forecasting and climate simulations. *Geoscientific Model Development*, 11(10):4043–4068, 2018. doi:10.5194/gmd-11-4043-2018. URL https://gmd.copernicus.org/articles/11/4043/2018/.

Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. *Proceedings of the 1968 23rd ACM national conference*, pages 517–524, 1968. doi:10.1145/800186.810616.

Michael Steiner, Wolfgang Peters, I. Luijkx, Stephan Henne, Huilin Chen, Simon Hammer, and Dominik Brunner. European ch_4 inversions with icon-art coupled to the carbontracker data assimilation shell. *Atmospheric Chemistry and Physics*, 24(4):2759–2782, 2024. doi:10.5194/acp-24-2759-2024. URL https://doi.org/10.5194/acp-24-2759-2024.

Joël Thanwerdas, Antoine Berchet, Lionel Constantin, Aki Tsuruta, Michael Steiner, Friedemann Reum, Stephan Henne, and Dominik Brunner. Improving the ensemble square root filter (ensrf) in the community inversion framework: a case study with icon-art 2024.01. *Geoscientific Model Development*, 18(5):1505–1544, 2025. doi:10.5194/gmd-18-1505-2025. URL https://doi.org/10.5194/gmd-18-1505-2025.

Michael Weimer, Jennifer Schröter, Johannes Eckstein, Konrad Deetz, Marco Neumaier, Garlich Fischbeck, Lu Hu, Dylan B. Millet, Daniel Rieger, Heike Vogel, Bernhard Vogel, Thomas Reddmann, Oliver Kirner, Roland Ruhnke, and Peter Braesicke. An emission module for icon-art 2.0: implementation and simulations of acetone. *Geoscientific Model Development*, 10(6): 2471–2494, 2017. doi:10.5194/gmd-10-2471-2017. URL https://doi.org/10.5194/gmd-10-2471-2017.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eigenständigkeitserklärung

Die unterzeichnete Eigenständigkeitserklärung ist Bestandteil jeder während des Studiums verfassten schriftlichen Arbeit. Eine der folgenden zwei Optionen ist **in Absprache mit der verantwortlichen Betreuungsperson** verbindlich auszuwählen:

☐ Ich erkläre hiermit, dass ich die vorliegende Arbeit eigenverantwortlich verfasst habe, namentlich, dass mir niemand beim Verfassen der Arbeit geholfen hat. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge der Betreuungsperson. Es wurden keine Technologien der generativen künstlichen Intelligenz[1] verwendet.

☒ Ich erkläre hiermit, dass ich die vorliegende Arbeit eigenverantwortlich verfasst habe. Dabei habe ich nur die erlaubten Hilfsmittel verwendet, darunter sprachliche und inhaltliche Korrekturvorschläge der Betreuungsperson sowie Technologien der generativen künstlichen Intelligenz. Deren Einsatz und Kennzeichnung ist mit der Betreuungsperson abgesprochen.

Titel der Arbeit:

OPIN: An online data interpolator for the ICON-ART atmospheric transport model using the ComIn Interface

Verfasst von:
Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	**Vorname(n):**
Hug	Zeno Maximilian

Ich bestätige mit meiner Unterschrift:
- Ich habe mich an die Regeln des «Zitierleitfadens» gehalten.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu und vollständig dokumentiert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

Ich nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Eigenständigkeit überprüft werden kann.

Ort, Datum	**Unterschrift(en)**
Dübendorf, 27.08.2025	Zeno Hug

Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich. Durch die Unterschriften bürgen sie grundsätzlich gemeinsam für den gesamten Inhalt dieser schriftlichen Arbeit.

[1] Für weitere Informationen konsultieren Sie bitte die Webseiten der ETH Zürich, bspw. https://ethz.ch/de/die-eth-zuerich/lehre/ai-in-education.html und https://library.ethz.ch/forschen-und-publizieren/Wissenschaftliches-Schreiben-an-der-ETH-Zuerich.html (Änderungen vorbehalten).