Astronomy and Computing 55 (2026) 101069

Contents lists available at ScienceDirect

Astronomy and Computing

journal homepage: www.elsevier.com/locate/ascom

VisIVO interactive high performance visualization☆

E. Sciacca[a],*, N. Tuccari[a,b], U. Arshad[c], F. Pitari[d], G. Muscianisi[d], C. Carbone[e], F. Vitello[a], B. Montalto[a]

[a] *INAF Astrophysical Observatory of Catania, Via Santa Sofia 78, Catania, 95123, Italy*
[b] *University of Catania, Department of Mathematics and Informatics, Viale Andrea Doria 6, Catania, 95125, Italy*
[c] *University of L'Aquila, Department of Information Engineering, Computer Science and Mathematics, Via Vetoio, L'Aquila, 67100, Italy*
[d] *Cineca, Via Magnanelli 6/3, Casalecchio di Reno, Bologna, 40033, Italy*
[e] *Istituto di Astrofisica Spaziale e Fisica cosmica Milano, Via Alfonso Corti 12, Milano, 20133, Italy*

ARTICLE INFO

Keywords:
Interactive visualization
VisIVO
Jupyter notebooks
High-performance computing
Cloud computing

ABSTRACT

The exponential growth of data in Astrophysics and Cosmology demands scalable computational infrastructures coupled with interactive and reproducible visualization tools. This paper presents the integration of the VisIVO scientific visualization framework within the InterActive Computing (IAC) service at Cineca, providing real-time, GPU-accelerated visualization directly on HPC resources through a browser-based Jupyter interface. By developing dedicated Python wrappers and a custom Jupyter kernel, VisIVO can now be executed seamlessly within interactive notebooks, eliminating traditional command-line complexities and enabling immediate visualization on HPC compute nodes. Additionally, the incorporation of RESTful APIs based on the Flask framework enables the remote execution of VisIVO operations through lightweight web services, further abstracting backend complexity and enhancing interoperability with external applications. The framework improves user accessibility, fosters reproducibility, and supports high-throughput data exploration across large-scale astrophysical simulations. The system has been validated through representative use cases, including visual analysis of cosmological simulations produced with the GADGET code, demonstrating its scalability, robustness, and capacity to support interactive scientific discovery on HPC infrastructures.

1. Introduction

Modern astrophysical and cosmological research operates at data scales that increasingly challenge traditional computational workflows. Observational surveys and large-scale simulations routinely generate petabyte-level datasets, requiring not only powerful high-performance computing (HPC) systems but also advanced tools for real-time visualization, exploration, and analysis. Interactive visualization of such data is essential for understanding complex phenomena such as cosmic structure formation, galaxy clustering, and matter distribution in the Universe. However, achieving this interactivity within HPC environments remains non-trivial, as most systems rely on batch scheduling and command-line access that hinder real-time feedback and reproducibility.

VisIVO[1] (Visualization Interface for the Virtual Observatory) is a mature suite of tools for high-performance, multidimensional data analysis and visualization in astrophysics. Initially developed for distributed infrastructures such as grids and clouds (Becciani et al., 2015a), and later adapted for the European Open Science Cloud (EOSC) through containerized science gateways (Sciacca et al., 2022, 2023), VisIVO has evolved toward supporting interactive and portable workflows. The latest step in this evolution involves its integration with the InterActive Computing (IAC[2]) service (Alam et al., 2021) currently up and running on the Galileo 100 cluster (G100[3]) at Cineca, an HPC service designed to provide browser-based, on-demand access to compute nodes through Jupyter notebooks.

The IAC service transforms traditional batch-oriented HPC access into a fully interactive, web-based environment. With IAC, users can launch notebook sessions directly on compute nodes equipped with GPUs and high-speed interconnects. This paradigm supports interactive data analysis, real-time visualization, and simulation steering, addressing long-standing challenges in accessibility and user engagement. By coupling VisIVO with IAC, we enable researchers to perform 3D

☆ This article is part of a Special issue entitled: 'HPC in Cosmology and Astrophysics' published in Astronomy and Computing.
* Corresponding author.
E-mail address: eva.sciacca@inaf.it (E. Sciacca).
[1] VisIVO, https://visivo.readthedocs.io/.
[2] IAC, https://jupyter.g100.cineca.it/.
[3] Galileo 100 infrastructure: https://www.hpc.cineca.it/systems/hardware/galileo100/.

https://doi.org/10.1016/j.ascom.2026.101069
Received 30 October 2025; Received in revised form 28 December 2025; Accepted 23 January 2026
Available online 5 February 2026

visualization workflows interactively within an HPC context, without the need for command-line configuration or manual data transfer. This integration brings HPC-level scalability to Jupyter-based environments, fostering reproducible and user-friendly scientific workflows.

Our contribution focuses on embedding VisIVO's visualization capabilities within the IAC infrastructure through the creation of Python wrappers and a dedicated Jupyter kernel, which automatically load modules, manage environment variables, and invoke the underlying C++ binaries transparently. This integration not only simplifies user interaction but also ensures consistent execution across distributed resources. Furthermore, by leveraging HPC technologies such as Spack and Ansible, we achieve a reproducible and modular deployment of VisIVO within the IAC ecosystem, paving the way for generalized science gateways capable of coupling domain-specific visualization tools with HPC backends.

In parallel, we have initiated the development of REST-based services that extend VisIVO's functionalities beyond the notebook environment. These APIs, implemented using lightweight Python frameworks, will allow remote execution of visualization tasks, enabling programmatic and web-accessible interfaces for large-scale data rendering and analysis. This ongoing work complements the interactive IAC integration by promoting interoperability, automation, and future cloud–HPC hybrid deployment.

The paper is organized as follows: Section 2 reviews related efforts in interactive and HPC-enabled visualization; Section 3 describes the modules of VisIVO Server and its overall architecture; Section 4 introduces the InterActive Computing (IAC) service; Section 5 details the methodology for integrating VisIVO into the IAC framework while Section 6 reports the development of the REST-based visualization APIs; Section 7 presents representative astrophysical use cases, including AI-assisted Super Resolution simulations. Finally, Section 8 summarizes conclusions and outlines future extensions.

2. Related works

Scientific visualization at scale has undergone significant evolution, particularly within Astrophysics and Cosmology, where petascale data volumes generated by simulations and surveys require real-time analysis and visual insight. Existing tools and platforms have laid important foundations, but few offer an integrated, interactive HPC experience that couples real-time data analysis with notebook-based visualization in a reproducible way.

Among the earliest efforts to facilitate visualization at scale with HPC, VisIt (Childs et al., 2012), ParaView (Ahrens et al., 2005) and CARTA (Comrie et al., 2025) have provided robust platforms for parallel and remote visualization across distributed systems. While all of them support client–server architectures suitable for HPC environments, they require expert setup and lack native Jupyter integration, limiting accessibility for domain scientists.

Interactive web-based solutions have gained traction through science gateways and cloud-oriented platforms. For example, Becciani et al. (2015b) demonstrated how gateways can expose advanced astrophysical analysis tools like VisIVO to researchers via web portals. Building on this, Sciacca et al. (2023) introduced the NEANIAS Visualisation Gateway, which integrated VisIVO into the European Open Science Cloud (EOSC) via containerized services and Jupyter-based user interfaces. This work established the feasibility of deploying astrophysical visualization capabilities in federated cloud infrastructures, while enabling reproducibility and portability across platforms.

However, these gateways have typically lacked deep integration with HPC schedulers or real-time session management, relying on batch-executed backends or coarse-grained interactivity.

With the increasing adoption of Jupyter Notebooks as de facto interfaces for reproducible scientific workflows, several projects have explored integrating them with supercomputing resources. Kluyver et al. (2016) proposed Jupyter as a front-end to scientific computing environments, while modern initiatives like the Fenix infrastructure (Alam et al., 2021) extended this model for neuroscience and brain research. These platforms demonstrated the feasibility of wrapping HPC systems behind notebook-based web portals, but largely emphasized static analysis over GPU-accelerated, interactive visualization workflows.

Dykes et al. (2021) proposed a generic framework to enable browser-based interaction with running HPC codes, including visualization with Splotch (Rivi et al., 2014), using lightweight web technologies. While that approach primarily addresses accessibility and interactive control through application-agnostic web interfaces, the technical focus differs from the solution presented here. Our work introduces a tighter, HPC-native integration of a domain-specific visualization framework, embedding VisIVO directly within the InterActive Computing (IAC) service through a dedicated Jupyter kernel and Python wrappers. This approach avoids external middleware and application refactoring, enabling GPU-accelerated visualization to run natively on compute nodes within reproducible, notebook-driven workflows. As such, our contribution emphasizes a deeper integration with HPC execution, scheduling, and deployment mechanisms, extending beyond remote interaction toward fully integrated interactive visualization on supercomputing infrastructures.

Compared to previous work of Sciacca et al. (2023), the developments presented in this paper eliminate the need for manual module loading, SSH-based file retrieval, or CLI-based rendering. Furthermore, it provides real-time visualization capabilities through VisIVO Viewer and facilitates multi-step processing pipelines via pre-defined, scriptable notebooks.

Additionally, while Spack (Gamblin et al., 2015) and Ansible (Hochstein and Moser, 2017) have been widely adopted for deployment automation in HPC software stacks, their use here for integrating Python and C++-based tools into seamless notebook environments underscores a new level of composability. This work transforms static CLI workflows into reproducible, interactive sessions, thus advancing the reproducibility and usability of scientific visualization in astrophysics.

In contrast to more generic frameworks, this approach is domain-specific, targeting astrophysical simulations (e.g., GADGET) and workflows centered on density rendering and cosmological structure analysis. It also opens the door to REST-based services and serverless visualization backends that can further abstract HPC complexity while maintaining performances.

3. VisIVO server

VisIVO Server is a suite of software tools designed to create customized 3D visualizations from astrophysical data, supporting large-scale datasets without fixed limits on dimensionality.

Its main modules available via Command Line Interface (CLI) are as follows:

- VisIVO Importer: converts user-supplied datasets into VisIVO Binary Tables (VBT), an efficient internal data representation. It supports various formats, including the general purpose data formats such as ASCII or CSV or tailored astronomical data formats such as FITS, HDF5, GADGET and more.
- VisIVO Filter: processes VBTs to apply data transformations, filtering, and other operations to prepare datasets for visualization.
- VisIVO Viewer: generates interactive 3D visualizations from VBTs, allowing users to explore and analyze their data effectively.

A typical usage of VisIVO Server involves at least three steps for data preparation, processing and visualization (see, e.g., Fig. 1). For sample VisIVO commands please see Section 7.

Data preparation. Use VisIVO Importer to convert your dataset into a VBT.

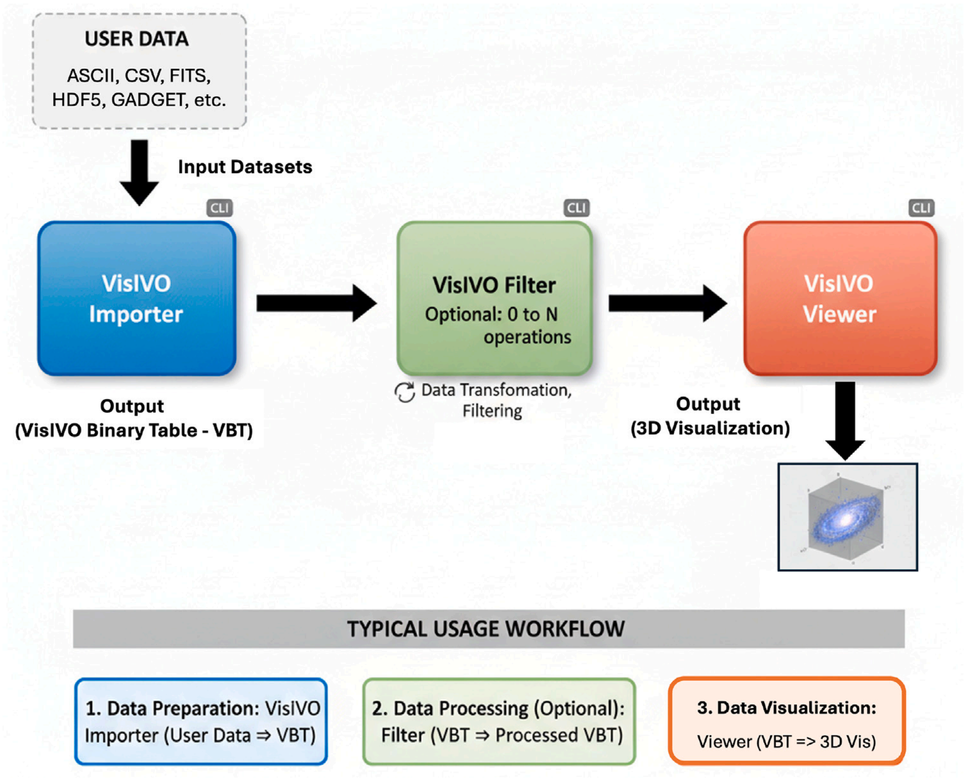

USER DATA
ASCII, CSV, FITS, HDF5, GADGET, etc.
Input Datasets
CLI
VisIVO Importer
Output (VisIVO Binary Table - VBT)
VisIVO Filter
Optional: 0 to N operations
Data Transfomation, Filtering
VisIVO Viewer
Output (3D Visualization)
TYPICAL USAGE WORKFLOW
1. Data Preparation: VisIVO Importer (User Data ⇒ VBT)
2. Data Processing (Optional): Filter (VBT ⇒ Processed VBT)
3. Data Visualization: Viewer (VBT => 3D Vis)

Fig. 1. VisIVO Server modular architecture and basic workflow involving importer, filter and view operations.

Data processing (optional). Apply one or more VisIVO Filter operations to perform data transformations or filtering on the VBT as needed.

Data visualization. Utilize one or more VisIVO Viewer renderings to create 3D visualizations from the (processed) VBT.

This modular workflow allows for efficient handling and visualization of complex astrophysical datasets.

4. InterActive Computing service (IAC)

As scientific workflows grow more complex and interactive, they require computing capabilities beyond traditional batch processing. To address this requirement, the Fenix[4] infrastructure makes InterActive Computing (IAC) a core service in its federated portfolio, tailored to the neuroscience community served by the Human Brain Project[5] (HBP) and its successor, EBRAINS.[6] IAC allows researchers to interact dynamically with data and applications, supporting rapid feedback and more efficient scientific progress. IAC in Fenix was implemented and managed by Cineca and E4 using E4's ICE4HPC suite,[7] and it is currently operational on Cineca's Galileo 100[8] cluster.

Two primary benefits motivate for an interactive approach to HPC resources.

1. Users gain direct, browser-based access to HPC compute nodes at https://jupyter.g100.cineca.it. This method differs significantly from the traditional HPC user experience, which typically involves *SSH* (Secure Shell) access to a shared login node and submitting jobs via a batch system (a system for queuing and scheduling computation tasks). Because the legacy workflow is command-line-only, graphical visualization of results is limited, whereas the interactive web interface makes it seamless.
2. The interactive approach to HPC resources allows near-instant access, enabling immediate interaction through a web browser. This contrasts with the batch model, where job start times are inherently unpredictable for users. It enables on-the-fly interaction with running workflows (Jette and Wickberg, 2023), real-time monitoring of resource usage, and visualization of intermediate results to guide subsequent workflow steps. A workflow here refers to a sequence of tasks or computations performed to analyze data.

Fig. 2 outlines the service architecture, which comprises three components:

1. First, a frontend virtual machine (VM) exposes the website to the public internet, kept separate from cluster login nodes for security. After credentials and two-factor authentication, it presents a form with dropdowns to request HPC resource allocations.
2. Next, the virtual machine (VM), which has been granted controlled, exceptional access to the Slurm controller, submits a job to a dedicated partition using the requested resources. That job launches a Jupyter-based (Kluyver et al., 2016) server, which is tunneled back to the VM for external access.
3. Finally, near-instantaneous access is achieved by exploiting a dedicated Slurm partition ("backend nodes" in Fig. 2) where CPU oversubscribe is allowed; this means that if the partition is fully allocated, multiple users may share the same CPU. GPUs are not oversubscribed and are allocated exclusively.

The user gets an interactive HPC session end-to-end, as shown in the user-interaction UML diagram of Fig. 3. From a local browser, the user submits credentials and resource requests (1) to a cloud IAC frontend, which keeps the web surface outside the cluster for security. The frontend forwards the resource request (2–3) to the cluster's Slurm

[4] Fenix, https://fenix-ri.eu/.
[5] Human Brain Project, https://humanbrainproject.eu/.
[6] EBRAINS, https://ebrains.eu/.
[7] ICE4HPC, https://www.e4company.com/en/ice4hpc/.
[8] Galileo 100 infrastructure: https://www.hpc.cineca.it/systems/hardware/galileo100/.

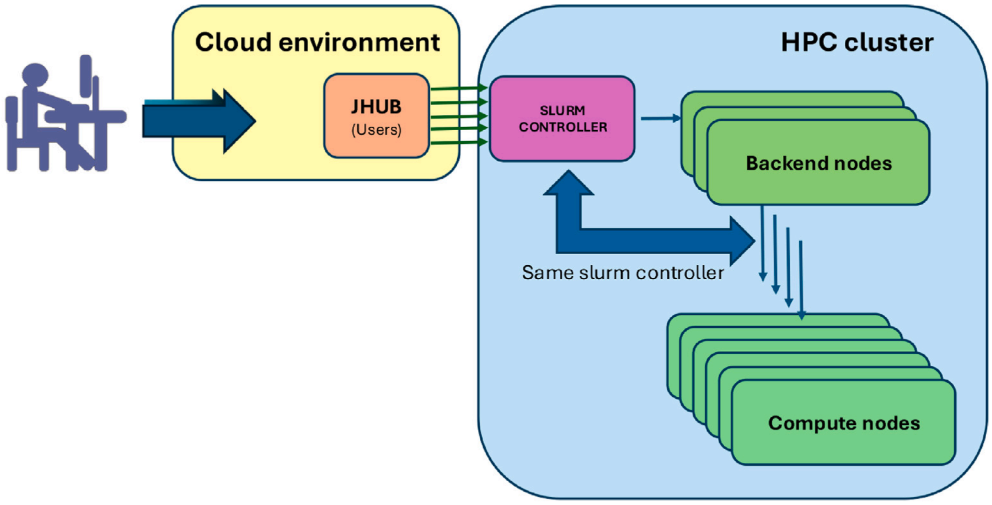


Fig. 2. InterActive Computing service implementation at Cineca.

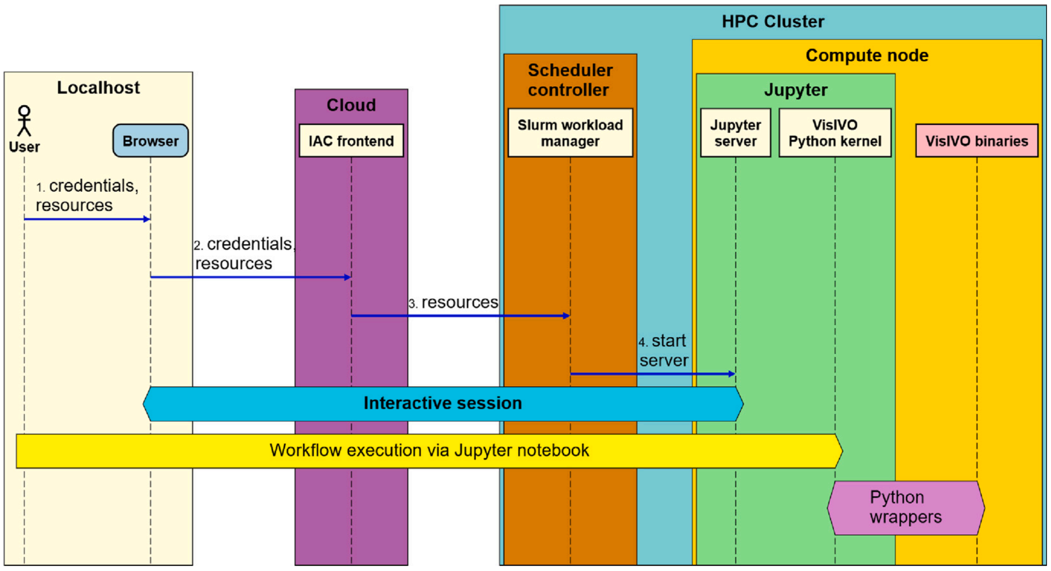


Fig. 3. InterActive Computing VisIVO service implementation at Cineca.

scheduler controller, which allocates a compute node in a dedicated partition and starts a Jupyter server there (4). That Jupyter server exposes a VisIVO-enabled Python kernel on the compute node, where Python wrappers call the underlying VisIVO binaries. The user then works in a continuous interactive session—executing notebooks, monitoring resources in real time, and visualizing intermediate results, while the heavy computation runs directly on the HPC node.

The IAC is being built on Jupyter as it aligns naturally with interactive HPC. Jupyter supports a single-user server–client model, offers a comprehensive plugin ecosystem (such as monitoring tools and jupyter-server-proxy for exposing additional HTTP services through the dashboard), and is widely recognized by users, which eases the introductory experience with HPC. Security is integral to the architecture: all sensitive orchestrations are executed on a dedicated virtual machine within Cineca's cloud, while HPC operations are run as standard SLURM jobs in user space, eliminating the need for root privileges. Furthermore, two-factor authentication secures logins, and continuous monitoring, vulnerability assessments, and penetration testing are performed to proactively identify issues.

5. Methodology

Integrating a VisIVO-specific kernel into the InterActive Computing (IAC) service streamlines workflows and removes manual module loads. Previously, G100 users logged in via *ssh* and used the command line to load VisIVO before they could work. With that setup, accessing PNG outputs required extra steps and they were not easily retrievable while working within the cluster. In this work, we instead provide a dedicated VisIVO Jupyter kernel within the IAC interface. The kernel is preconfigured: users start an IAC session and select "VisIVO", and the module loads automatically. VisIVO commands then run out of the box, and the resulting PNG files are saved directly to the user's session and are immediately accessible and viewable through the IAC web interface. Overall, this integration reduces friction and improves efficiency, letting users focus on scientific visualization rather than command-line setup. This framework works despite VisIVO not being a Python code, thus not natively supported for Jupyter integration; in the following Section we will describe how such integration has been achieved.

5.1. VisIVO wrappers

VisIVO is written in C/C++, so building it produces standalone binaries. Although Jupyter does not natively embed arbitrary binaries in the user interface, it does support interactions through custom Python environments. To bridge this gap, we created a Python wrapper[9] that connects Python with the VisIVO binaries, enabling VisIVO commands to run from within a Python session.

Our implementation primarily uses Python's standard library; specifically, we call the VisIVO binaries through the *subprocess* module.

The Python wrapper program structures its functions to facilitate the management of the three primary VisIVO commands: VisIVOImporter, VisIVOFilter, and VisIVOViewer. To clarify, each function is designed to be command-specific while also managing shared tasks such as invoking the CLI, recording logs, and validating function arguments against each command's permitted options. All three functions follow the same pattern (with `importer` shown below): options from the VisIVOImporter binary are exposed as the function's parameters.

```python
def importer(input_file, *flags, mpi=False, tasks=None,
    fformat=None, out=None, volume=None, compx=None,
   [...]):
```

Subsequently, each item is processed to determine its variable type using an `options` dictionary that forwards these options to the VisIVO commands.

We chose this approach over more compact designs to keep the VisIVO CLI as hidden from users as possible. Accordingly, we expose all options as explicit function parameters rather than bundling them into a single optional list of tuples.

The `_corefunction` operates through several key steps. Initially, it verifies the type of option values received from the outer function. Subsequently, boolean variables are converted into options without assigned values, while all other variables are converted into options with assigned values. Finally, add the input file to complete the VisIVO command. The command can run with MPI or in serial mode, depending on whether the variables `mpi` and `tasks` are set. More details on the code can be found in Appendix A.

The VisIVO kernel includes the *Pillow*[10] module to improve image rendering, with particular benefits for PNG formats which, via the command line, is inefficient and often necessitates supplementary utilities or manual file access. As a result, users generate images faster and with less hassle.

With *Pillow* integrated, images are displayed directly within the VisIVO Python session, removing unnecessary intermediate steps. This enhancement streamlines the workflow by allowing immediate image display in Jupyter notebooks and facilitating efficient inspection of graphical results.

To exploit such a possibility, we added a `plot` function in the VisIVO Python wrapper to allow us to easily plot histograms obtained from the VisIVO Filter *statistic* operation. The function relies on `matplotlib` to plot data inside Jupyter notebooks. Notably, this is a clear example of how embedding VisIVO capabilities in a broader environment like Jupyter largely extended its possibility, since it can now rely on a wide spectrum of tools without the need of implementing them internally in the source code. An example of the resulting interactive logarithmic histogram generated by the `plot` function, useful to explore the distribution of a scalar quantity, is shown in Fig. 4.

[9] VisIVO Python Wrapper: https://github.com/VisIVOLab/VisIVOPythonWrapper.

[10] Pillow library: https://pillow.readthedocs.io/.

5.2. IAC deployment

The environment setup process consists of four main steps:

1. compilation of VisIVO with the Spack module
2. Use *Ansible* to create the *conda* environment on the IAC backend nodes.
3. Install the Python wrappers from the GitHub repository.
4. Customize the Jupyter *ipykernel*.

First, install VisIVO system-wide by compiling it with Spack (Gamblin et al., 2015). A Spack module to enforce the OSMesa backend across graphical dependencies, particularly VTK and GLEW. This ensures correct image generation in IAC: without an active windowing system (only the web front end runs), we rely on off-screen rendering.

Next, provision a *conda* environment using *Ansible* (Hochstein and Moser, 2017). This environment hosts the Python wrappers described in Section 5. Although the wrappers only depend on the Python standard library, adding convenience packages for visualization (e.g., Pillow, NumPy-based libraries, Matplotlib) can improve the user experience. Python offers a clear benefit over the traditional VisIVO setup. Users can analyze their data using both VisIVO and standard Python tools in the same environment. They can also display images instantly using *IPython* display features within the Jupyter web interface.

Ansible playbooks facilitate the deployment of the VisIVO virtual environment and enable parallel software installation on each computational node's local storage within the Slurm partition allocated to the InterActive Computing service. This approach is preferred over installations that utilize the cluster's parallel file system, as multiple tests with the latter demonstrated reduced performance during the service login phase. The performance degradation is attributed to the parallel file system's inefficient handling of numerous small files, which occurs when the Jupyter server initialization procedure reads multiple backend conda environments. Deployment uses an Ansible inventory that explicitly lists all backend nodes participating in the service and is launched from the system's login node (which serves as the Ansible control node).

Subsequently, the ipykernel interface responsible for initializing the VisIVO kernel in the Jupyter environment is modified to execute a bash script as a prolog. This modification enables loading the Spack module that contains the VisIVO installation and sets the required environment variables so that the VisIVO module is available to be launched as a Notebook or Console environment as shown in Fig. 5.

6. Automated pipeline using VisIVO

A further use case we want to address with our codebase is the one related to pipelines. Automated workflows are essential for modern scientific projects with high data throughput. Many typical examples for these use cases can be found in fields like weather forecasting, tsunami alerts, or data management from satellites: in all these cases multiple servers will interact among them cascading, triggering HPC computations automatically at some point. To start such interaction, there is no standard approach as it is strongly tightly coupled to the specific access policies of the HPC site involved. Nevertheless, in a different use case like the IAC implementation mentioned in Section 5.2, we faced the same issue and this was solved via Cloud-HPC interaction. Coupling cloud and HPC infrastructure, as seen for the IAC case, allows to expose web-based services which are not typically allowed on HPC clusters. A possible solution for the pipeline use case, then, is to implement standard web REST APIs within HPC workflows, thereby coupling cloud and HPC to enable secure, scalable, and reproducible automation from submission through result retrieval. REST APIs allow a general standard approach for web services interactions, making the service agnostic to the client used; for instance requests can be sent both via command line (e.g. curl command) or via any web frontend, without any change needed on the server side.

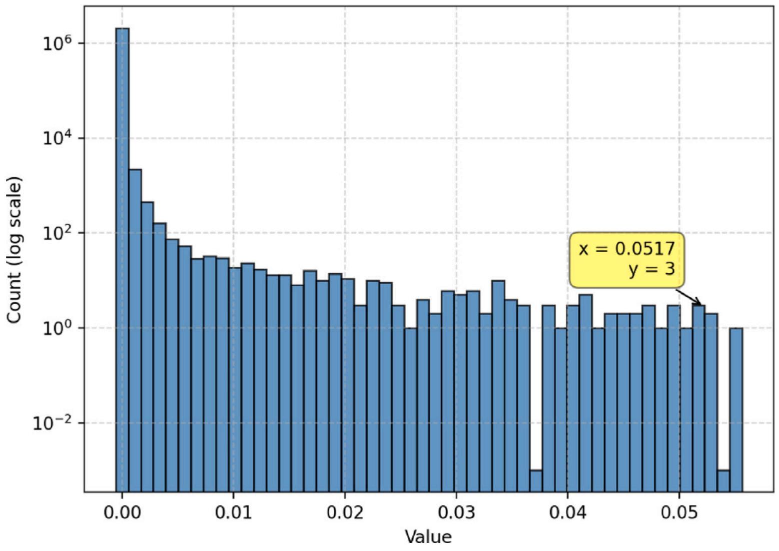
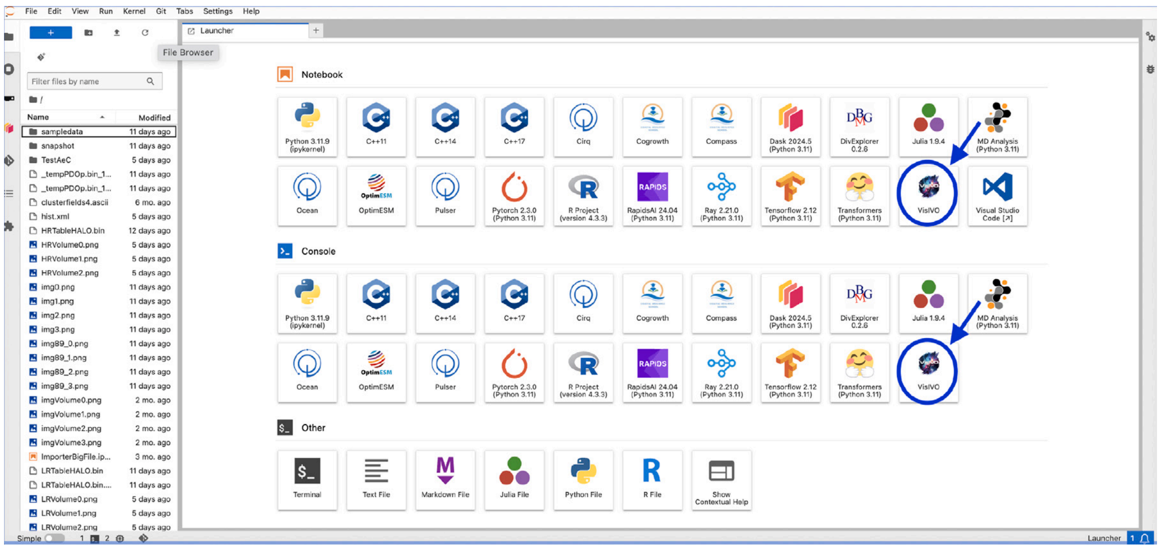


Fig. 4. Interactive log count histogram produced using VisIVO Filter *statistic* that illustrates the distribution of a scalar quantity, e.g. the particles' density, using logarithmic counts for exploratory data analysis.



Filter files by name

/

Name	Modified
sampledata	11 days ago
snapshot	11 days ago
TestAeC	5 days ago
_tempPDOp.bin_1...	11 days ago
_tempPDOp.bin_1...	11 days ago
clusterfields4.ascii	6 mo. ago
hist.xml	5 days ago
HRTableHALO.bin	12 days ago
HRVolume0.png	5 days ago
HRVolume1.png	5 days ago
HRVolume2.png	5 days ago
img0.png	11 days ago
img1.png	11 days ago
img2.png	11 days ago
img3.png	11 days ago
img89_0.png	11 days ago
img89_1.png	11 days ago
img89_2.png	11 days ago
img89_3.png	11 days ago
imgVolume0.png	2 mo. ago
imgVolume1.png	2 mo. ago
imgVolume2.png	2 mo. ago
imgVolume3.png	2 mo. ago
ImporterBigFile.ip...	3 mo. ago
LRTableHALO.bin	11 days ago
LRTableHALO.bin....	11 days ago
LRVolume0.png	5 days ago
LRVolume1.png	5 days ago
LRVolume2.png	5 days ago

Fig. 5. VisIVO Server module, highlighted in blue, available from the Jupyter interface ready to be launched as a Notebook or Console environment.

Similar needs were present in many use cases such as the Terabit project (Zuccolo et al., 2025a,b) for rapid earthquake-related simulations and the GAIA (Prusti et al., 2016) mission for post-processing large-scale astronomical observations: these pipelines integrate and coordinate the flow of data from remote servers, such as collecting real-time measurements from telescopes or seismic sensors, across several interconnected steps; however, no general approach exists for such use cases, thus implementing REST APIs might be a candidate standard solution for future similar needs. To prototype such approach, we used our VisIVO codebases to simulate a pipeline in which simulations data are automatically post-processed with VisIVO via REST API requests.

6.1. Deployment as a service

In order to achieve the goal of integration in pipelines via REST API, we integrated Flask (Relan, 2019) in our original code, i.e. the VisIVO Python wrappers described in Appendix A. The choice of Flask is well suited to the implementation for two main reasons:

- Flask requires very few lines to easily expose a web server from a Python code, and it handles requests and responds according to app's instructions. The developer can choose the network address and port for the server. In development mode, the app automatically reloads whenever the developer makes code changes, so updates are reflected instantly, and it also displays useful error messages with an interactive debugger.

python
```python
app = Flask(__name__)
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5000, debug=True)
```

- It provides functions decorators to implement REST API endpoints; this fits perfectly with our starting code, which relies on three functions for its three main capabilities. In other words, we needed just to decorate such functions with Flask decorators to create three endpoints for the running server;

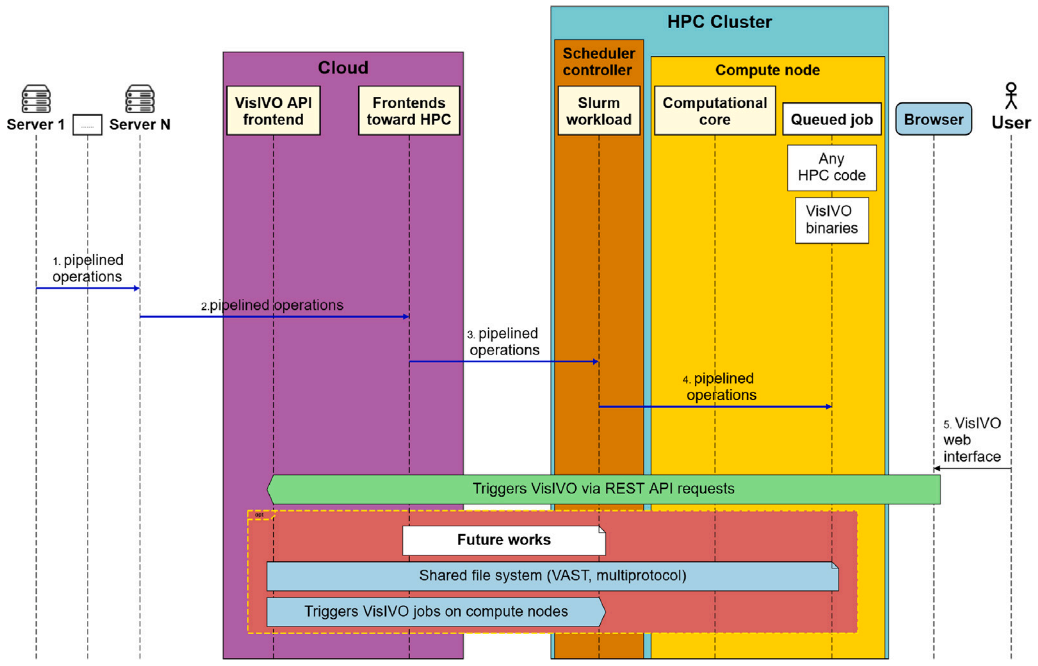


Fig. 6. VisIVO REST API design diagram. The red box represents the future works reported in Section 8.

```python
@app.route('/some_endpoint', methods=['POST', 'GET'])
def some_endpoint():
```

We hosted our REST API service on a VM in the OpenStack-based Cineca infrastructure, ADA cloud.[11] Then, we installed VisIVO building a Docker container dedicated to the service; this is not a mandatory approach, and we might have installed a non-containerized version of VisIVO, but being on the cloud this is a more convenient and portable solution. In addition to the container, we used a dedicated Python virtual environment to run the server. This sequence provides an isolated, lightweight backend layer to support the application's REST API implementation.

6.2. *Pipeline workflow example*

We exposed the Flask API endpoints on a dedicated virtual machine within Cineca's cloud infrastructure, ensuring a secure environment as detailed in Section 6.1. On this VM, each request is validated, the service logs record relevant information, and translates the request into a function call for the VisIVO Python wrapper (which in turn is translated into VisIVO command line calls, as described in Appendix A). In the current implementation on our VM the client submits a payload (for example, to /importer), the service immediately validates the inputs, invokes the corresponding VisIVO command line, and returns a response containing retrieval information such as images, outputs, or derived files upon completion. The ultimate purpose is to create a language-independent interface, leveraging an established general approach such as REST APIs to hide the complexity of the underlying software architecture; for instance, this would allow triggering VisIVO in any automated pipelines via any REST client, or to integrate the VisIVO capabilities in any web interface (examples are shown in Fig. 6).

The current prototype is still running on the cloud side and has not yet exploited HPC capabilities. This might be extended in the future with a more general approach involving job submission directly on the cluster, as well as a shared file system among the VM and the cluster, in order not to move data. This latter scenario is also sketched in Fig. 6, and more details are given in Section 8 when describing future works.

7. Use case applications

7.1. *Super resolution visualization*

In this Section we demonstrate the integration of the VisIVO Server with the IAC through a real use case scenario where machine learning and visualization are involved (Montalto et al., in preparation).

In this use case, we visually compare the low-resolution (LR), high-resolution (HR), and AI-generated Super Resolution (SR) snapshots of cosmological simulations. The SR results are obtained using an independently developed model, consistent with previous studies that demonstrated the applicability of Super Resolution methods to cosmological data (Li et al., 2021). Once the LR-SR mapping is learned, it can be applied to computationally less expensive low-resolution simulations to approximate the results that would have been obtained from a finer (and costlier) simulation.

Here we employ datasets from the DEMNUni (Dark Energy and Massive Neutrino Universe) cosmological simulations, designed to study the massive neutrinos influences on observables, in particular on galaxies, clusterers, cosmic voids, and cosmic microwave background (CMB) surveys (Carbone et al., 2016). To this scope, large volume simulations were produced, to capture perturbations on extended cosmic scales, while maintaining a high mass resolution, allowing the study of non-linear phenomena on small scale due to free-streaming neutrinos.

7.1.1. *Super resolution workflow*

In this workflow, we import the HR cosmological simulation snapshots in GADGET format, along with their downsampled LR counterparts and the AI-generated SR versions, all containing the dark matter particles' positions. The importing of the simulation is executed using MPI/OpenMP because the VisIVO GADGET importer has been recently implemented to run of HPC infrastructures scaling on multi-nodes and multi-cores when available. An example command is shown below for running on 2 cores and 4 nodes:

[11] ADA cloud, https://www.hpc.cineca.it/systems/hardware/ada-cloud/.

```bash
export OMP_NUM_THREADS=2
mpirun --np 4 VisIVOImporter --fformat gadget --out
     NewTable --file snapdir/snap_data.0
```

Then the HALO particles in VBT format are filtered to extract a sub-box given a geometry description.

```bash
VisIVOFilter --op extraction --geometry geometry.txt  --
     out sub-box.bin --file NewTableHALO.bin
```

We create a volume from the particles position using the *pointdistribute* filter which produces a density field distributed and divided for the volume voxels on a sample 64 × 64 × 64 resolution volume.

```bash
VisIVOFilter --op pointdistribute --resolution 64 64 64
     --points POS_X POS_Y POS_Z --out densityvolume.bin
     --file sub-box.bin
```

The final volume is then visualized using a volume rendering algorithm.

```bash
VisIVOViewer --volume --vrendering --vrenderingfield
     Constant --color --colortable volren_glow --showlut
      --out img --file densityvolume.bin
```

7.1.2. Super resolution interactive notebook

The VisIVO module is available to be launched as a Notebook or Console environment, see Fig. 5. The notebook implementing the Super Resolution workflow described in the previous Section 7.1.1 is shown in Fig. 7.

The interactive notebooks require the VisIVO wrappers module to be loaded.

```python
import visivo
```

The notebook then executes the following commands:

```python
visivo.importer("/sr_data/srdata.0", fformat="gadget
 ", out="/testSR/SR", mpi=True, tasks=4)
visivo.filter("/testSR/SRHALO.bin", op="extraction",
 geometry="/testSR/geometry.txt", out="/testSR/
 sub_box_100_SR")
visivo.filter("/testSR/sub_box_100_SR.bin", op="
 pointdistribute", resolution="384 384 384", points="
 POS_X POS_Y POS_Z", out="/testSR/densityvolume.bin")
visivo.viewer("/testSR/densityvolume.bin", volume=
 True, vrendering=True, vrenderingfield="Constant",
 color=True, colortable="volren_glow", autorange=
 True, showlut=True, showaxes=True, showbox=True,
 out="img")
```

The first command uses the VisIVO Importer; in this case we convert the GADGET snapshot, containing the cosmological particles' positions in a 3D environment, into a VBT. The importer can also be executed in parallel using MPI exploiting the capabilities of the cluster.

The second command applies a filter to the generated VBT that extracts a sub-box, this is done to be able to visualize specific structures contained in the sub-box. The command expects as input a geometry.txt file, which defines the parameters of the extraction box. In this case, the file has the following format:

```
POS_X  500.0
POS_Y  500.0
POS_Z  950.0
BOX     50.0
```

These values specify the center of the sub-box at coordinates (500.0, 500.0, 950.0) and a half-side length of 50. Consequently, the extracted region is a cube with a total side length of 100.

The third command then applies the pointproperty filter to compute the particle densities on a 384 × 384 × 384 resolution volume. The command expects the desired resolution to be specified as a space-separated list of three integers corresponding to the X, Y, and Z dimensions, and it also requires the names of the three fields that define the particle coordinates in the VBT, in this case POS_X, POS_Y, and POS_Z.

Finally the fourth command invokes the VisIVO Viewer module to produce a 3D visualization of the computed density field. The volume and vrendering options enable volume rendering, while vrenderingfield="Constant" specifies the name of the field within the volume VBT. In this case, Constant is the default field name generated by the pointdistribute filter. Enabling the autorange option activates a VisIVO Viewer feature that automatically adjusts the color range to ensure optimal visualization. The options showlut, showaxes, and showbox display the color legend, coordinate axes, and bounding box, respectively.

These operations were executed on three sets of data: Low Resolution, High Resolution and Super Resolution. The resulting volume renderings are shown and compared in Fig. 8.

7.2. REST API use case

In this Section we show the integration of the VisIVO Server with the VisIVO deployment as a service REST APIs described in Section 6 through two sample workflows first reported in Sciacca et al. (2024) and briefly summarized hereafter.

7.2.1. VisIVO workflow examples

The first workflow imports a sample TXT file containing cosmological particles' positions employing the following command:

```bash
   VisIVOImporter --fformat ascii clusterfields4.ascii
```

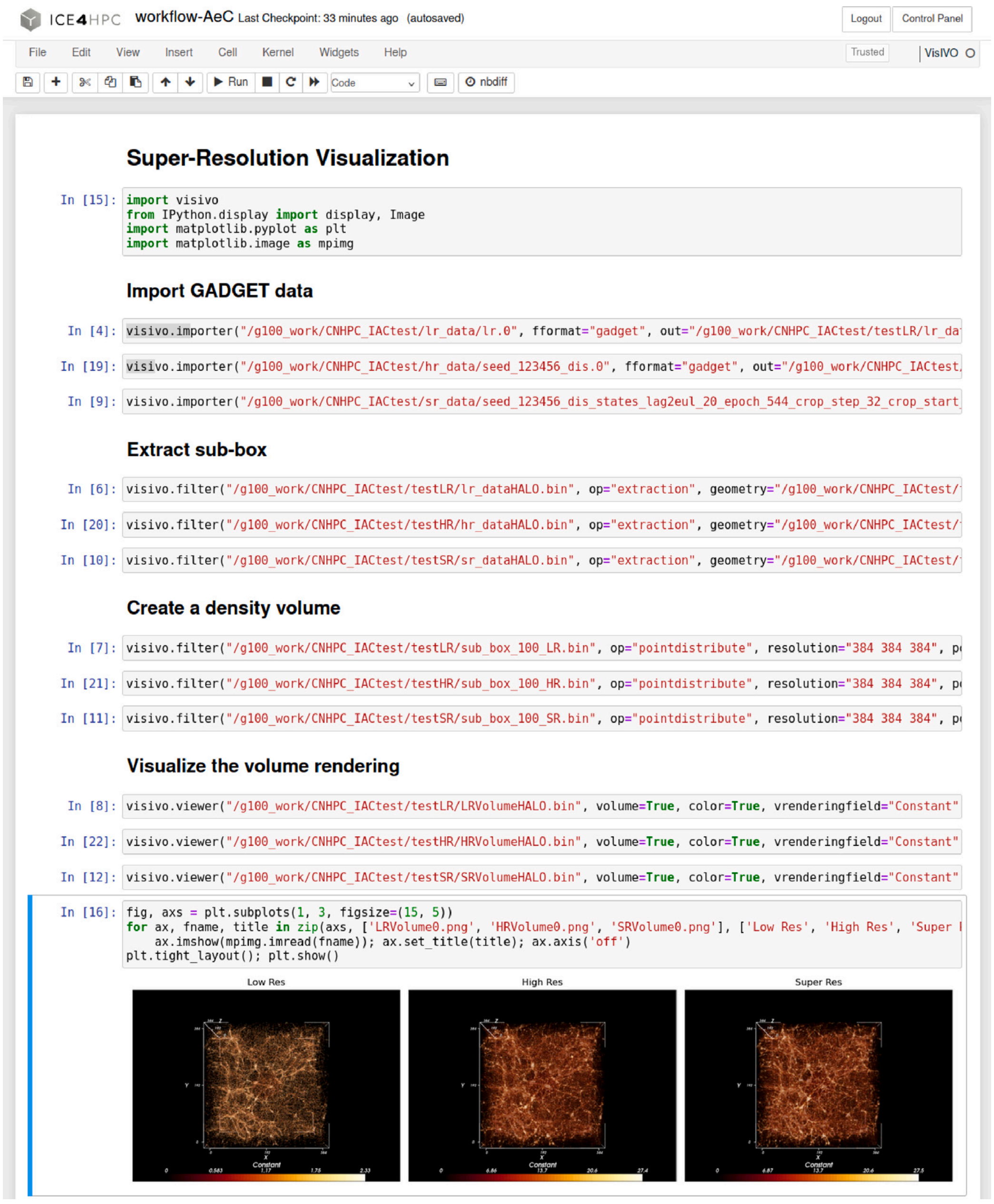
and visualizes it using a data points rendering:

```bash
VisIVOViewer -x X -y Y -z Z --scale --glyphs pixel
     VisIVOServerBinary.bin
```

The second workflow imports a sample snapshot of a cosmological simulation in GADGET format containing cosmological particles' positions of the GAS, HALO and STARS. The importing of the simulation is executed using MPI/OpenMP because the VisIVO GADGET importer has been recently implemented to run on HPC infrastructures scaling on multi-nodes and multi-cores when available. This is a sample command:

```bash
mpirun --np 4 VisIVOImporter --fformat gadget --out
     NewTable --file snapdir/snap_091.0
```

The HALO particles in VBT format are then filtered to create a volume from the particles position using the *pointdistribute* filter which produces a density field distributed and divided for the volume voxels.



workflow-AeC Last Checkpoint: 33 minutes ago (autosaved)

Super-Resolution Visualization

```
In [15]: import visivo
         from IPython.display import display, Image
         import matplotlib.pyplot as plt
         import matplotlib.image as mpimg
```

Import GADGET data

```
In [4]: visivo.importer("/g100_work/CNHPC_IACtest/lr_data/lr.0", fformat="gadget", out="/g100_work/CNHPC_IACtest/testLR/lr_da

In [19]: visivo.importer("/g100_work/CNHPC_IACtest/hr_data/seed_123456_dis.0", fformat="gadget", out="/g100_work/CNHPC_IACtest,

In [9]: visivo.importer("/g100_work/CNHPC_IACtest/sr_data/seed_123456_dis_states_lag2eul_20_epoch_544_crop_step_32_crop_start_
```

Extract sub-box

```
In [6]: visivo.filter("/g100_work/CNHPC_IACtest/testLR/lr_dataHALO.bin", op="extraction", geometry="/g100_work/CNHPC_IACtest/

In [20]: visivo.filter("/g100_work/CNHPC_IACtest/testHR/hr_dataHALO.bin", op="extraction", geometry="/g100_work/CNHPC_IACtest/

In [10]: visivo.filter("/g100_work/CNHPC_IACtest/testSR/sr_dataHALO.bin", op="extraction", geometry="/g100_work/CNHPC_IACtest/
```

Create a density volume

```
In [7]: visivo.filter("/g100_work/CNHPC_IACtest/testLR/sub_box_100_LR.bin", op="pointdistribute", resolution="384 384 384", p

In [21]: visivo.filter("/g100_work/CNHPC_IACtest/testHR/sub_box_100_HR.bin", op="pointdistribute", resolution="384 384 384", p

In [11]: visivo.filter("/g100_work/CNHPC_IACtest/testSR/sub_box_100_SR.bin", op="pointdistribute", resolution="384 384 384", p
```

Visualize the volume rendering

```
In [8]: visivo.viewer("/g100_work/CNHPC_IACtest/testLR/LRVolumeHALO.bin", volume=True, color=True, vrenderingfield="Constant"

In [22]: visivo.viewer("/g100_work/CNHPC_IACtest/testHR/HRVolumeHALO.bin", volume=True, color=True, vrenderingfield="Constant"

In [12]: visivo.viewer("/g100_work/CNHPC_IACtest/testSR/SRVolumeHALO.bin", volume=True, color=True, vrenderingfield="Constant"

In [16]: fig, axs = plt.subplots(1, 3, figsize=(15, 5))
         for ax, fname, title in zip(axs, ['LRVolume0.png', 'HRVolume0.png', 'SRVolume0.png'], ['Low Res', 'High Res', 'Super 
             ax.imshow(mpimg.imread(fname)); ax.set_title(title); ax.axis('off')
         plt.tight_layout(); plt.show()
```

Fig. 7. Jupyter notebook running VisIVO via web browser on a compute node of Galileo 100 cluster.

```
bash

VisIVOFilter --op pointdistribute --resolution 64 64 64
    --points POS_X POS_Y POS_Z --out densityvolume.bin
    --file NewTableHALO.bin
```

The final volume is then visualized using a volume rendering algorithm.

```
bash

VisIVOViewer --volume --vrendering --vrenderingfield
    Constant --color --colortable volren_glow --showlut
     --out img --file densityvolume.bin
```

7.2.2. *REST-API usage*

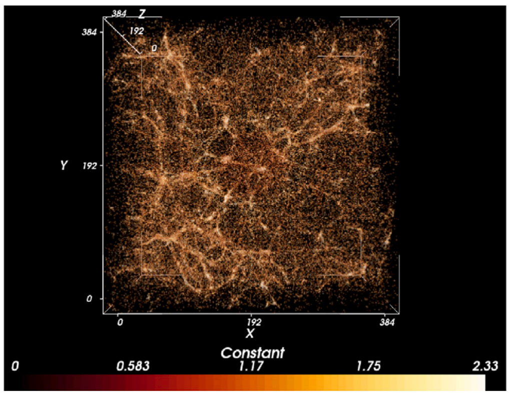
In this Section we demonstrate how to approach the same workflows using HTTP requests via the REST API exposed in Section 6. For the sake of clarity we will present here both the JSON syntax that can be used for instance via "curl" command and some HTML samples to trigger VisIVO via HTTP request in a web interface.

Here below we will use 192.168.0.89:5000 just as examples for ip and port, and /path/for/input/file/ for the filepath the user chose:

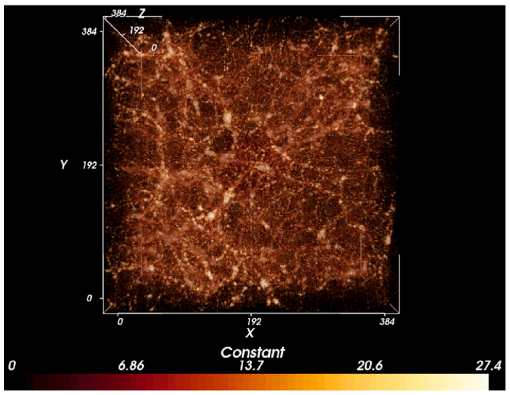
The JSON payloads for the first workflow of Section 7.2.1 would include the VisIVO parameters in the body:

```
curl -X POST http://192.168.0.89:5000/importer \
  -H "Content-Type: application/json" \
  -d '{
    "path": "/path/for/clusterfields4.ascii/file/",
    "fformat": "ascii"
  }'
```

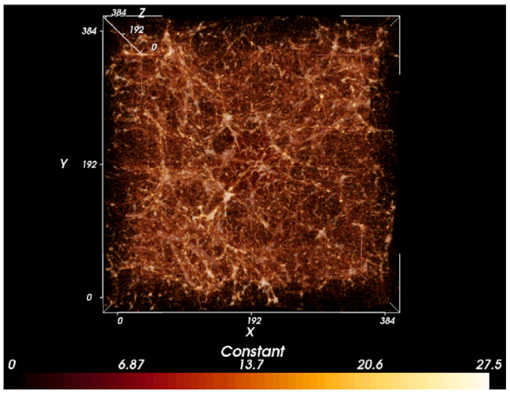
whereas it would look like the following in a basic HTML implementation:



(a) Low resolution data rendering.



(b) High resolution data rendering.



(c) Super resolution data rendering.

Fig. 8. Volume Rendering comparison between multiple datasets: Low Resolution, High Resolution, Super Resolution.

```
<form action="http://192.168.0.89:5000/importer"
 ↪  method="post">
  <input name="path"
   ↪  value="/path/for/clusterfields4.ascii">
  <input name="fformat" value="ascii">
  <button type="submit">Importer</button>
</form>
```

For the viewer endpoint the JSON payload would look like the following:

```
curl -X POST http://192.168.0.89:5000/viewer \
  -H "Content-Type: application/json" \
  -d '{
    "path": "/path/for/VisIVOServerBinary.bin/file",
    "x": "X",
    "y": "Y",
    "z": "Z",
    "scale": true,
    "glyphs": "pixel"
  }'
```

HTML implementation for the viewer:

```
<form action="http://192.168.0.89:5000/viewer"
 ↪  method="post">
  <input name="path"
   ↪  value="/path/for/VisIVOServerBinary.bin">
  <input name="x" value="X">
  <input name="y" value="Y">
  <input name="z" value="Z">
  <input name="scale" value="true">
  <input name="glyphs" value="pixel">
  <button type="submit">Viewer</button>
</form>
```

The curl and HTML code for the second workflow are shown in Appendix B.

8. Conclusions and future works

This work has presented the integration of the VisIVO scientific visualization framework within the InterActive Computing (IAC) service at Cineca, establishing a new paradigm for interactive, reproducible, and scalable visualization directly on HPC infrastructures. Through this integration, VisIVO — traditionally operated through command-line interfaces and batch processing — has been transformed into a fully interactive, browser-based visualization environment, enabling researchers to execute complex 3D visual analyses directly from Jupyter notebooks connected to GPU-enabled compute nodes.

The development of Python wrappers and a dedicated Jupyter kernel has significantly simplified user interaction, allowing the execution of VisIVO commands without manual configuration or module loading. These enhancements not only streamline the visualization workflow but also ensure transparent access to HPC resources, accelerating exploratory analysis and reducing the technical barriers to entry for scientific users. Additionally, the incorporation of RESTful APIs based on the Flask framework enable the remote execution of VisIVO operations through lightweight web services, further abstracting backend complexity and enhancing interoperability with external applications. The approach demonstrates how traditional visualization tools can evolve into HPC-native, user-centric systems, merging performance, accessibility, and reproducibility in a single environment.

The presented framework was validated through representative astrophysical use cases, including visualization of large-scale cosmological simulations from the DEMNUni project and AI-assisted Super Resolution datasets. The results confirm the flexibility, versatility, and robustness of the system in managing and visualizing complex data, bridging the gap between batch-based supercomputing and real-time, interactive science.

Future work will focus on exploiting the REST API capabilities together with the VAST multi-protocol shared storage, in order to let the frontend access the data produced on the cluster without moving the data; such framework might also exploit the object storage protocol guaranteed by VAST to access the data, making the overall setup even more flexible. In addition, authentication methods have to be added to the REST API frontend, and the best way would probably be to use token authentication methods to the Python server. A big step forward will be possible when an incoming multi-protocol shared storage system between cloud infrastructure and HPC clusters is in place. A storage like this, based on the VAST technology (Opeolu et al., 2023), is going to be up and running at Cineca: When it is in place, we will replace every simulation currently running on the cloud with job submissions on the HPC cluster, and the results from the HPC simulation will be exposed directly by the VM thanks to the shared storage (see Fig. 6).

CRediT authorship contribution statement

E. Sciacca: Writing – original draft, Supervision, Software, Methodology, Funding acquisition, Conceptualization. **N. Tuccari:** Writing – review & editing, Visualization, Software. **U. Arshad:** Writing – original draft, Software. **F. Pitari:** Writing – original draft, Supervision, Methodology, Conceptualization. **G. Muscianisi:** Writing – review & editing, Supervision. **C. Carbone:** Writing – review & editing, Supervision, Data curation. **F. Vitello:** Writing – review & editing, Software. **B. Montalto:** Writing – review & editing, Data curation.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work, the authors made use of OpenAI's ChatGPT and Google's Gemini for paraphrasing, rewording, and checking grammar and spelling. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The work is supported by the EuroHPC JU under grant agreement No 101093441 and by the Spoke 1 "FutureHPC & BigData" and the Spoke 3 "Astrophysics and Cosmos Observations" of the ICSC – Centro Nazionale di Ricerca in High Performance Computing, Big Data and Quantum Computing, funded by NextGenerationEU. The DEMNUni simulations were carried out in the framework of "The Dark Energy and Massive-Neutrino Universe" project, using the Tier-0 IBM BG/Q Fermi machine and the Tier-0 Intel OmniPath Cluster Marconi-A1 of the Centro Interuniversitario del Nord-Est per il Calcolo Elettronico (Cineca). C. Carbone acknowledges a generous CPU and storage allocation by the Italian Super-Computing Resource Allocation (ISCRA) as well as from the coordination of the "Accordo Quadro MoU per lo svolgimento di attività congiunta di ricerca Nuove frontiere in Astrofisica: HPC e Data Exploration di nuova generazione", together with storage from INFN-CNAF and INAF-IA2.

Appendix A. VisIVO wrapper implementation

The Python wrapper program structures its functions to facilitate the management of the three primary VisIVO commands: VisIVOImporter, VisIVOFilter, and VisIVOViewer. All three functions follow the same pattern (with `importer` shown below): options from the VisIVOImporter binary are exposed as the function's parameters.

```python
def importer(input_file, *flags, mpi=False, tasks=None,
    fformat=None, out=None, volume=None, compx=None,
   [...]):
```

Subsequently, each item was processed to determine its variable type using an `options` dictionary.

```python
    options = {
    "fformat": (fformat, str),
    "out": (out, str),
    "volume": (volume, str),
    "compx": (compx, float),
    "compy": (compy, float),
    "compz": (compz, float),
    "sizex": (sizex, float),
    [...]
```

The dictionary is provided to a function that forwards these options to the VisIVO commands.

```python
command = "VisIVOImporter"
_corefunction(command, input_file, *flags, mpi=mpi,
   tasks=tasks, options=options)
```

We chose this approach over more compact designs to keep the VisIVO CLI as hidden from users as possible. Accordingly, we expose all options as explicit function parameters rather than bundling them into a single optional list of tuples.

The `_corefunction` operates through several key steps. Initially, it verifies the type of option values received from the outer function.

```python
worked_options = {}
if options:
    for key, (val, expected_type) in options.items():
        if val is not None:
            if not isinstance(val, expected_type):
                raise TypeError(f"'{key}' must be of type {
    expected_type.__name__}")
            if expected_type is bool:
                if val: # add only if True
                    worked_options[key] = val
            else:
                worked_options[key] = val
```

Subsequently, boolean variables are converted into options without assigned values, while all other variables are converted into options with assigned values.

```python
for option, value in worked_options.items():
    if isinstance(value, (list, tuple)):
        command.append(f"--{option}")
        command.extend(map(str, value))
    elif value and type(value) != bool:
        command.append(f"--{option}")
        command.append(str(value))
    else:
        if value != False:
            command.append(f"--{option}")
```

Finally, add the input file to complete the VisIVO command:

```python
   command.append(input_file)
```

You can run this command with MPI or in serial mode, depending on whether the variables `mpi` and `tasks` are set.

```python
if mpi:
    mpi_command = ["mpirun", "-n", str(tasks)] + command
    logging.debug(f"Running MPI command: {mpi_command}"
    )
    result = subprocess.run(mpi_command, capture_output
    =True, text=True)
else:
    logging.debug(f"Running command: {command}")
    result = subprocess.run(command, capture_output=
    True, text=True)
```

Some smaller code sections have been left out to keep things clear and easy to follow.

Appendix B. Additional workflow

In the second workflow described in Section 7.2.1, the JSON payloads should have the VisIVO parameters included in the body.

```
curl -X POST http://192.168.0.89:5000/importer \
  -H "Content-Type: application/json" \
  -d '{
    "path": "/path/to/snapdir/snap_091.0",
    "fformat": "gadget",
    "out": "NewTable",
    "mpi": true,
    "np": 4
  }'
```

In contrast, the HTML approach would appear as follows:

```
<form action="http://192.168.0.89:5000/importer"
 ↪  method="post">
  <input name="path"
   ↪  value="/path/to/snapdir/snap_091.0">
  <input name="fformat" value="gadget">
  <input name="out" value="NewTable">
  <input type="hidden" name="mpi" value="true">
  <input type="number" name="np" value="4">
  <button type="submit">Importer</button>
</form>
```

For the second step this would result as follows:

```
curl -X POST http://192.168.0.89:5000/filter \
  -H "Content-Type: application/json" \
  -d '{
    "path": "/path/to/NewTableHALO.bin/file",
    "op": "pointdistribute",
    "resolution": "64 64 64",
    "points": "POS_X POS_Y POS_Z",
    "out": "densityvolume.bin"
  }'
```

```
<form action="http://192.168.0.89:5000/filter"
 ↪  method="post">
  <input name="path"
   ↪  value="/path/to/NewTableHALO.bin">
  <input name="op"         value="pointdistribute">
  <input name="resolution" value="64 64 64">
  <input name="points"     value="POS_X POS_Y POS_Z">
  <input name="out"        value="densityvolume.bin">
  <button type="submit">Filter</button>
</form>
```

For the third step, this would look like this:

```
curl -X POST http://192.168.0.89:5000/viewer \
  -H "Content-Type: application/json" \
  -d '{
    "path": "/path/to/densityvolume.bin/file",
    "volume": true,
    "vrendering": true,
    "vrenderingfield": "Constant",
    "color": true,
    "colortable": "volren_glow",
    "showlut": true,
    "out": "img"
  }'
```

```
<form action="http://192.168.0.89:5000/viewer"
 ↪  method="post">
  <input name="path"
   ↪  value="/absolute/path/to/densityvolume.bin">
  <input type="checkbox" name="volume" value="true">
  <input type="checkbox" name="vrendering" value="true">
  <input name="vrenderingfield" value="Constant">
  <input type="checkbox" name="color" value="true">
  <input name="colortable" value="volren_glow">
  <input type="checkbox" name="showlut" value="true">
  <input name="out" value="img">
  <button type="submit">Viewer</button>
</form>
```

References

Ahrens, J., Geveci, B., Law, C., 2005. ParaView: An end-user tool for large data visualization. In: Visualization Handbook. Elsevier, pp. 717–731.

Alam, S., Bartolome, J., Bassini, S., Carpene, M., Cestari, M., Combeau, F., Girona, S., Gorini, S., Fiameni, G., Hagemeier, B., et al., 2021. Fenix: Distributed e-infrastructure services for EBRAINS. In: Amunts, K., Grandinetti, L., Lippert, T., Petkov, N. (Eds.), Brain-Inspired Computing. Springer International Publishing, pp. 81–89. doi:10.1007/978-3-030-70710-8_7.

Becciani, U., Sciacca, E., Costa, A., Massimino, P., Pistagna, C., Riggi, S., Vitello, F., Petta, C., Bandieramonte, M., Krokos, M., 2015a. Science gateway technologies for the astrophysics community. Concurr. Comput.: Pract. Exp. 27 (2), 306–327.

Becciani, U., Sciacca, E., Costa, A., Massimino, P., Pistagna, C., Riggi, S., Vitello, F., Petta, C., Bandieramonte, M., Krokos, M., 2015b. Science gateway technologies for the astrophysics community. Concurr. Comput.: Pract. Exp. 27 (2), 306–327. doi:10.1002/cpe.3280.

Carbone, C., Petkova, M., Dolag, K., 2016. DEMNUni: ISW, rees-sciama, and weak-lensing in the presence of massive neutrinos. J. Cosmol. Astropart. Phys. 2016 (07), 034.

Childs, H., Brugger, E., Whitlock, B., Meredith, J., Ahern, S., Pugmire, D., Biagas, K., Miller, M., Harrison, C., Weber, G.H., Krishnan, H., Fogal, T., Sanderson, A., Garth, C., Bethel, E.W., Camp, D., Rübel, O., Durant, M., Favre, J.M., Navrátil, P., 2012. Visit: An End-User tool for visualizing and analyzing very large data. In: High Performance Visualization–Enabling Extreme-Scale Scientific Insight. pp. 357–372. doi:10.1201/b12985.

Comrie, A., Wang, K.-S., Hwang, Y.-H., Pińska, A., Harris, P., Raul-Omar, C., Hou, K.-C., Aikema, D., Chiang, C.-C., Ming-Yi, L., Huang, P.-S., Gao, Z.-K., Simmonds, R., 2025. CARTA: The cube analysis and rendering tool for astronomy. doi:10.5281/zenodo.17050846.

Dykes, T.J., Varetto, U., Gheller, C., Krokos, M., 2021. Real-time web-based remote interaction with active HPC applications. In: International Conference on Information Visualization Theory and Applications. SciTePress, pp. 88–100.

Gamblin, T., LeGendre, M., Collette, M.R., Lee, G.L., Moody, A., De Supinski, B.R., Futral, S., 2015. The spack package manager: bringing order to HPC software chaos. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–12.

Hochstein, L., Moser, R., 2017. Ansible: Up and running: Automating configuration management and deployment the easy way. " O'Reilly Media, Inc.".

Jette, M.A., Wickberg, T., 2023. Architecture of the slurm workload manager. In: Workshop on Job Scheduling Strategies for Parallel Processing. Springer, pp. 3–23.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B.E., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J.B., Grout, J., Corlay, S., et al., 2016. Jupyter notebooks.

Li, Y., Ni, Y., Croft, R.A., Di Matteo, T., Bird, S., Feng, Y., 2021. AI-assisted superresolution cosmological simulations. Proc. Natl. Acad. Sci. 118 (19), e2022038118.

Montalto, B., Sciacca, E., Vitello, F., Carbone, C., 2026. AI-Assisted Super-Resolution-Simulations: application to the DEMNUni- Cov N-Body simulations. Astron. Comput. in preparation.

Opeolu, V., Engelbrecht, H., Hu, S.-Y., Marais, C., 2023. VAST: A decentralized open-source publish/subscribe architecture. In: Proceedings of the 14th Conference on ACM Multimedia Systems. pp. 423–429.

Prusti, T., De Bruijne, J., Brown, A.G., Vallenari, A., Babusiaux, C., Bailer-Jones, C., Bastian, U., Biermann, M., Evans, D.W., Eyer, L., et al., 2016. The gaia mission. Astron. Astrophys. 595, A1.

Relan, K., 2019. Building REST APIs with flask. Building REST APIs with Flask.

Rivi, M., Gheller, C., Dykes, T., Krokos, M., Dolag, K., 2014. Gpu accelerated particle visualization with splotch. Astron. Comput. 5, 9–18.

Sciacca, E., Cesare, V., Tuccari, N., Vitello, F., Colonnelli, I., Carbone, C., Tramontana, E., Becciani, U., 2024. Toward a portable and reproducible High- performance visualization for astronomy & cosmology. doi:10.5281/zenodo.13858498.

Sciacca, E., Krokos, M., Bordiu, C., Brandt, C., Vitello, F., Bufano, F., Becciani, U., Raciti, M., Tudisco, G., Riggi, S., et al., 2022. Scientific visualization on the cloud: the NEANIAS services towards EOSC integration. J. Grid Comput. 20 (1), 7.

Sciacca, E., Raciti, M., Krokos, M., Kyd, B., et al., 2023. Science gateways in EOSC: The NEANIAS visualisation gateway. In: Proceedings of the 14th International Workshop on Science Gateways.

Zuccolo, E., Bolzon, G., Pitari, F., Monsalvo Franco, I.E., Scaini, C., Poggi, V., Smerzini, C., Salon, S., et al., 2025a. UrgentShake: An HPC system for near Real-Time Physics-Based ground shaking simulations to support emergency response efforts. In: EGU2025.

Zuccolo, E., Bolzon, G., Pitari, F., Rodríguez Muñoz, L., Scaini, C., Vanini, M., Poggi, V., Salon, S., 2025b. Advancing rapid response to earthquakes with tiered Physics-Based Ground-Shaking simulations: The UrgentShake system. Seismol. Res. Lett..