



RAPPORTI TECNICI INGV

MorphoVolc DEM Interface: piattaforma web-native per la morfometria e la stima volumetrica di edifici vulcanici



ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

519

**Direttore Responsabile**

Daniela VERSACE

Editor in Chief

Milena MORETTI

Editorial Board

Laura ALFONSI

Sofia BARANELLO

Carla BARNABA

Christian BIGNAMI

Simona CARANNANTE

Maria Grazia CIACCIO

Luca COCCHI

Luigi CUCCI

Lorenzo CUGLIARI

Elisabetta D'ANASTASIO

Alessia DI CAPRIO

Roberto DI MARTINO

Domenico DI MAURO

Domenico DORONZO

Filippo GRECO

Alessandro IAROCCI

Mario MATTIA

Daniele MELINI

Anna NARDI

Lucia NARDONE

Marco OLIVIERI

Arianna PESCI

Pierangelo ROMANO

Paola SBARRA

Davide SCAFIDI

Maurizio SOLDANI

Sara STOPPONI

Umberto TAMMARO

David ZULIANI

REGISTRAZIONE AL TRIBUNALE DI ROMA N.173 | 2014, 23 LUGLIO

© INGV Istituto Nazionale di Geofisica e Vulcanologia

Rappresentante legale: Presidente INGV

Sede: Via di Vigna Murata, 605 | Roma

Tutti i diritti riservati

www.ingv.it



ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

RAPPORTI TECNICI INGV

MorphoVolc DEM Interface: piattaforma web-native per la morfometria e la stima volumetrica di edifici vulcanici

MorphoVolc DEM Interface: a web-native platform for morphometry and volumetric estimation of volcanic edifices

Marco Solinas^{1,*}, Camilla Gentili², Massimo Musacchio¹, Malvina Silvestri¹, Maria Fabrizia Buongiorno¹, Sergio Falcone¹

¹INGV | Istituto Nazionale di Geofisica e Vulcanologia, Osservatorio Nazionale Terremoti

²Università degli Studi di Perugia

*Corresponding Author

Accettato 2 marzo 2026 | Accepted 2 March 2026

Come citare | How to cite Solinas, M., Gentili, C., Musacchio, M., Silvestri, M., Buongiorno, M.F., Falcone, S. (2026). MorphoVolc DEM Interface: piattaforma web-native per la morfometria e la stima volumetrica di edifici vulcanici. *Rapporti Tecnici INGV*, 519, 42 pp. <https://doi.org/10.13127/rpt/519>

In copertina Cratere Kerið, lago craterico vulcanico nell'Islanda meridionale (area di Selfoss). Foto Marco Solinas, 2022 | Cover Kerið crater, a volcanic crater lake in southern Iceland (Selfoss area). Marco Solinas's photo, 2022

INDICE

Riassunto	7
<i>Abstract</i>	7
Introduzione	8
1. MorphoVolc DEM Interface	9
1.1 Panoramica	9
1.2 Interfaccia utente	10
2. Architettura e servizi	10
2.1 FE-IN — Frontend input (user data upload and validation)	10
2.2 Derivative Computation (Python ⇄ Node; triggered from FE)	11
2.2.1 Script Structure and Computational Workflow	12
2.2.2 Headless execution and performance environment	13
2.2.3 Schema sintetico degli algoritmi (DEM ⇄ contorni ⇄ volumi)	14
2.3 Model Selection and Volumetric Computation (React ⇄ Node ⇄ Python)	14
2.3.1 Model Selection Logic and User Interaction	15
2.3.2 Volumetric computation workflow (Python scripts)	16
2.4 Volume Calculation & Results Preparation (FE-IN ⇄ FE-OUT)	17
2.4.1 Computation and Data Aggregation	18
2.4.2 Visualization and Export (React UI ⇄ Node)	18
2.5 Results & Export (FE-OUT)	19
2.5.1 Exportables & File Layout	20
2.5.2 Report Generation	20
2.5.3 Download Workflow	20
2.6 Application Routes and Artifacts Table	21
2.7 Dependencies and Environment	22
2.7.1 Runtime version	22
2.7.2 Frontend (React) — package.json highlights	22
2.7.3 Backend (Node/Express) — package.json highlights	22
2.7.4 Python pipeline — dependencies	22
2.7.5 Deployment and environment notes	23
3. Casi d’uso e prestazioni	23
3.1 Esempi applicativi (Miyakejima e Izu-Ōshima)	23
3.2 Minibenchmark e metodologia di misura	24
3.3 Confronto metodologico	24
4. Possibili applicazioni / integrazioni	24
4.1 Monitoraggio morfometrico operativo	25
4.2 Screening rapido su DEM 30 m	26
4.3 Batch via API / headless	26

4.4 Integrazione nei flussi di lavoro degli osservatori	26
4.5 Didattica e ricerca	27
4.6 Interoperabilità e roadmap di estensioni	27
5. Conclusioni e Sviluppi futuri	27
5.1 Sintesi dei risultati	27
5.2 Limiti attuali	28
5.3 Sviluppi futuri (roadmap)	28
5.4 Prospettive	29
Dichiarazione software	29
Disponibilità e prova dell'applicativo (v 0.1.1, Docker edition)	29
Bibliografia	30
Appendici	33

Riassunto

MorphoVolc DEM Interface è una piattaforma web open source progettata per semplificare e standardizzare l'analisi morfometrica di edifici vulcanici a partire da modelli digitali di elevazione. Il sistema calcola indicatori di forma e fornisce stime volumetriche dell'edificio (e, quando presente, della caldera) tramite modelli geometrici, rendendo il processo più trasparente e riproducibile.

L'architettura si basa su tre componenti principali: un'interfaccia utente sviluppata con il framework JavaScript React per caricare e gestire i dati dal browser; un servizio di orchestrazione basato su Node.js e sul framework Express; una pipeline di elaborazione in Python, eseguita in modalità non interattiva e basata su librerie per la lettura e l'analisi di dati raster e su librerie scientifiche, con un servizio web opzionale per l'esposizione come microservizio. La pipeline esegue i calcoli in modo automatico, senza dipendenze da database o grafica locale.

Il flusso di lavoro include la validazione del modello digitale di elevazione, il calcolo di derivate morfometriche (ombreggiatura, pendenza, curvatura, rugosità), la scelta del modello volumetrico (geometrie circolari o ellittiche con due modalità per la rappresentazione delle caldere) e l'esportazione dei risultati in formati geospaziali e tabellari e come immagini e report in formato portatile. Le formule analitiche adottate, basate principalmente sul modello di tronco di cono con eventuale cavità di caldera, sono accompagnate da una legenda che esplicita simboli, unità di misura e soglie operative (ad esempio 5% e 80% dell'elevazione massima), rendendo verificabili le assunzioni del modello.

I casi studio su Miyakejima e Izu-Ōshima mostrano tempi di elaborazione contenuti e risultati tracciabili, organizzati in pacchetti di output coerenti. La piattaforma supporta analisi operative su modelli digitali di elevazione a scala regionale (ad esempio con passo di griglia dell'ordine di decine di metri) e processi automatizzati anche in modalità batch tramite interfacce di programmazione.

Abstract

MorphoVolc DEM Interface is an open-source web platform designed to simplify and standardize the morphometric analysis of volcanic edifices based on digital elevation models. The system computes shape indicators and provides volumetric estimates of the edifice (and, when present, of the caldera) through geometric models, making the process more transparent and reproducible. The architecture is based on three main components: a user interface developed with the JavaScript framework React for uploading and managing data through the browser; an orchestration service built on Node.js and the Express framework; and a processing pipeline written in Python, executed in non-interactive mode and based on libraries for raster data handling and scientific computation, with an optional web service layer for exposure as a microservice. The pipeline performs calculations automatically, without reliance on databases or local graphical environments.

The workflow includes validation of the digital elevation model, computation of morphometric derivatives (hillshade, slope, curvature, roughness), selection of the volumetric model (circular or elliptical geometries with two representation modes for calderas), and export of results in geospatial and tabular formats as well as image and portable document reports. The analytical formulas adopted, primarily based on the truncated cone model with a potential caldera cavity, are accompanied by a legend explicitly defining symbols, measurement units, and operational thresholds (for example, 5% and 80% of maximum elevation), ensuring that the model assumptions are verifiable.

Case studies on Miyakejima and Izu-Ōshima demonstrate moderate processing times and fully traceable results, organized into coherent output packages. The platform supports operational analyses on regional-scale digital elevation models (for example, with grid spacing on the order of tens of meters) and automated batch processing through application programming interfaces.

Introduzione

Le morfologie vulcaniche derivano dall'interazione nel tempo di processi geologici e ambientali di tipo costruttivo e distruttivo. A causa del numero e della complessità di tali processi, i vulcani mostrano una grande varietà di forme e dimensioni. Nella maggior parte dei casi si tratta di rilievi costruiti dall'accumulo di prodotti eruttivi, ma possono presentarsi anche come strutture di escavazione (caldere, maar) o con un'espressione topografica debole o assente (ad esempio condotti fissurali o coltri piroclastiche estese). Le geometrie spaziano da coni regolari e simmetrici fino a catene e massicci vulcanici irregolari, con dimensioni che vanno da piccoli coni monogenetici alti poche decine di metri fino a edifici di migliaia di metri e decine di chilometri di diametro.

La morfometria vulcanica, ovvero la misura quantitativa della forma e delle dimensioni degli edifici vulcanici, consente di ricavare informazioni sui processi che ne hanno guidato la crescita e l'evoluzione. Sebbene forma e dimensione siano proprietà di base dei vulcani, non esiste ancora un database organico e aggiornato della morfometria vulcanica su scala globale.

La delimitazione dei contorni (base e caldera) e la stima dei volumi richiedono spesso operazioni manuali (scelta di soglie, editing dei poligoni, controlli visivi), mentre la mancanza di API documentate limita l'automazione e rende difficili confronti riproducibili tra dataset e gruppi di lavoro.

MorphoVolc DEM Interface nasce per colmare queste lacune, offrendo un sistema web-native open source che fornisce un flusso standardizzato e riproducibile: dai Modelli Digitali di Elevazione (DEM, acronimo invariabile nel testo) ai descrittori morfometrici e alle stime volumetriche degli edifici vulcanici. La versione rilasciata e utilizzabile è descritta nel box "Disponibilità e prova dell'applicativo" (v0.1.1, Docker edition) e nel repository GitHub associato. Il sistema si rivolge a osservatori vulcanologici, ricercatori in Scienze della Terra e docenti che necessitano di risultati rapidi, verificabili e confrontabili per analisi preliminari, studi comparativi o attività didattiche. L'interfaccia web, accessibile direttamente dal browser, ne consente l'uso anche a utenti non specialisti. L'API (Application Programming Interface) REST (Representational State Transfer) abilita l'elaborazione automatica e ripetuta di più DEM, integrandosi facilmente nei flussi di lavoro operativi. Il versioning dell'API è previsto negli sviluppi futuri (cfr. §5.3).

MorphoVolc DEM Interface si colloca nel filone degli strumenti di morfometria automatizzata (ad es. MORVOLC e NETVOLC), impiegati per la delimitazione e l'analisi degli edifici vulcanici anche in contesti di pericolosità morfologica e tsunami-genica [Zorn et al., 2022], e introduce un approccio nativo per il web con API e tracciabilità degli artefatti.

La Figura 1 sintetizza il flusso operativo di MorphoVolc DEM Interface: caricamento e validazione del DEM → calcolo delle derivate → selezione del modello → calcolo dei volumi → visualizzazione ed esportazione dei risultati. La figura riassume la sequenza algoritmica end-to-end e i punti di interazione utente (UI) rispetto alle esecuzioni python.

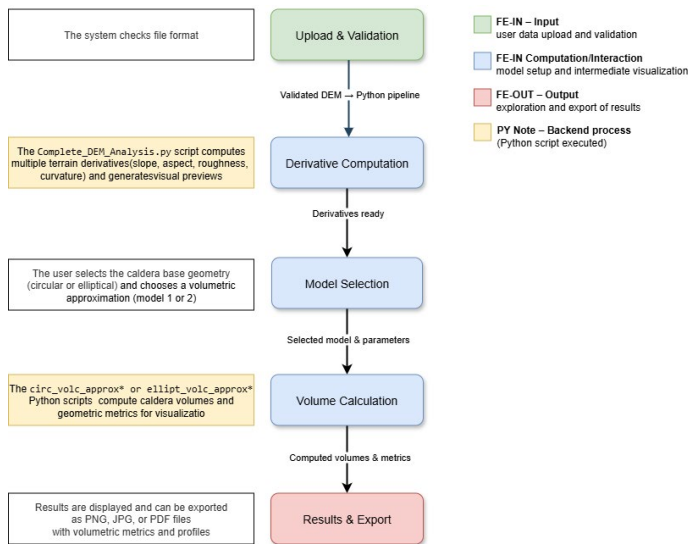


Figura 1 Schema generale del workflow end-to-end (FE-IN/FE-OUT) e dei processi Python (PY) in MorphoVolc DEM Interface: input DEM → derivate → selezione modello → volumi → risultati/export.

Figure 1 End-to-end workflow (FE-IN/FE-OUT) and Python backend processes (PY) in MorphoVolc DEM Interface: DEM input → derivatives → model selection → volumes → results/export.

1. MorphoVolc DEM Interface

1.1 Panoramica

MorphoVolc DEM Interface è un sistema nativo per il web dedicato all'analisi morfometrica e alla stima volumetrica di edifici vulcanici a partire da Digital Elevation Models (DEM). Nel presente Rapporto Tecnico si descrive la release v0.1.1, distribuita in forma dockerizzata ed eseguibile in locale tramite browser (cfr. box "Disponibilità e prova dell'applicativo").

L'architettura integra un frontend React (interfaccia utente), un servizio Node/Express (orchestrazione delle richieste) e una pipeline Python headless (geoprocessing e calcolo dei volumi), invocata come processo figlio dal servizio Node. Nello specifico:

- React è una libreria JavaScript per la costruzione di interfacce utente modulari e reattive; in MorphoVolc gestisce la logica di *frontend* e l'interazione con l'utente tramite componenti riutilizzabili.
- Node/Express costituisce il servizio intermedio lato server, basato su JavaScript, che riceve le richieste dal *frontend* e orchestra l'esecuzione dei processi Python, fungendo da ponte tra interfaccia e *pipeline* di calcolo.
- La pipeline Python è eseguita in modalità headless come processo figlio del servizio Node ed è responsabile dell'elaborazione numerica e della generazione dei prodotti (derivate, metriche, report). In alternativa, la pipeline può essere esposta come microservizio REST tramite FastAPI (cfr. §2.7.4).

Per garantire semplicità operativa e massima riproducibilità non viene utilizzato alcun database: il servizio espone rotte HTTP (REST locale) e scrive tutti gli artefatti (mappe, tabelle, report) su filesystem in cartelle strutturate per processId.

Il sistema è organizzato in moduli indipendenti che comunicano tra loro, secondo il modello delle architetture a servizi (SOA, *Service-Oriented Architecture*). Questo approccio—già adottato, ad esempio, da OpenTopography—consente di integrare in modo efficiente l'accesso a dati topografici ad alta risoluzione, gli strumenti di elaborazione e la scalabilità dell'infrastruttura, mantenendo buone prestazioni [Krishnan et al., 2011].

Nel seguito useremo derivate morfometriche per indicare i prodotti di base calcolati a partire dal DEM (ad es. pendenza, esposizione, curvature), impiegati per la delimitazione di base/sommità e per le stime volumetriche; i dettagli operativi sono riportati in §2.2.

1.2 Interfaccia utente

All'avvio del sistema, l'utente accede a una interfaccia web semplice e intuitiva (Figura 2), composta da due pannelli principali: sulla sinistra è presente una sezione descrittiva che riassume lo scopo dell'applicativo e le tipologie di analisi possibili, mentre sulla destra compare un box di caricamento DEM.

Quest'ultimo consente di selezionare manualmente il file o di utilizzare la modalità drag & drop, accettando raster DEM in formato GeoTIFF (.tif), con sistema di riferimento (CRS) definito tramite codice EPSG.

Dopo il caricamento, il sistema esegue soltanto i controlli preliminari essenziali che consistono nella presenza del file e il formato ammesso. L'avvio dell'elaborazione delle derivate avviene solo quando l'utente preme il pulsante "PROCESS DEM FILE" (per il dettaglio sul calcolo delle derivate si veda §2.2).

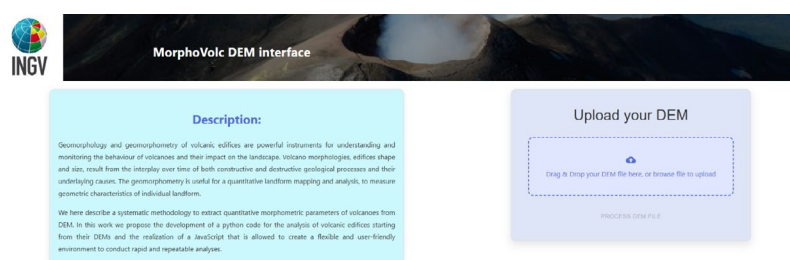


Figura 2 Interfaccia principale: sezione descrittiva (sinistra) e box di caricamento DEM (destra).

Figure 2 Main interface: descriptive section (left) and DEM upload box (right).

2. Architettura e servizi

2.1 FE-IN — *Frontend* input (user data upload and validation)

Nella release v0.1.1 l'intero stack applicativo (frontend, servizio Node/Express e pipeline Python) viene eseguito all'interno di un container Docker. L'utente interagisce esclusivamente tramite browser con una web UI locale esposta su <http://localhost:5000>, senza installazione manuale di Python/Node.js o dipendenze scientifiche sul sistema host.

Il blocco d'ingresso del frontend costituisce il punto d'ingresso operativo di *MorphoVolc DEM Interface* e ha la funzione di gestire il caricamento dei dati di input e la loro validazione preliminare. Attraverso il modulo di caricamento (*Upload Form*) l'utente può selezionare manualmente il file oppure utilizzare la modalità *drag & drop* all'interno del box dedicato sulla destra dell'interfaccia. Il sistema accetta raster DEM in formato GeoTIFF (.tif) che presentino un sistema di riferimento di coordinate (CRS) con un codice EPSG valido (es EPSG:4326), condizione necessaria per garantire la corretta georeferenziazione durante le elaborazioni successive. Questa fase iniziale è progettata per essere estremamente semplice: dopo il caricamento, l'interfaccia esegue soltanto verifiche di base (presenza del file, estensione e dimensioni coerenti) rimandando ogni altra operazione alla successiva fase di elaborazione. Solo quando l'utente preme il pulsante "PROCESS DEM FILE", il sistema avvia il flusso di elaborazione vero e proprio, che comprende la generazione delle derivate topografiche e la creazione dei primi artefatti di processo.

Il caricamento e la validazione sono gestiti dal form React integrato nel *frontend*. Al momento del click su "PROCESS DEM FILE", il modulo invia una richiesta POST /process al servizio Node/Express esposto localmente dal container, che riceve il file e i relativi metadati, genera

un identificativo univoco di processo (`processId`) e inizializza l'esecuzione Python in modalità *headless* ([Appendice C]). Il *frontend* conserva il valore di `processId` e interroga periodicamente lo stato dell'elaborazione tramite `GET /processStatus/{processId}`, aggiornando la barra di avanzamento e i messaggi di attesa o di errore ([Appendice A]).

La gestione dello stato dell'interfaccia si basa su un *hook* React dedicato (`useProcessStatus`), che implementa un *polling* con back-off lineare e l'aggiornamento progressivo della *UI* (*User Interface*) fino al completamento del *job*.

Dal lato server, Node/Express opera come *strato di orchestrazione* tra *frontend* e *pipeline* Python: riceve il DEM e i metadati, crea il `processId`, avvia lo script Python come child process con i parametri serializzati e, al termine, riceve la *callback* `POST /processComplete/{processId}`, che aggiorna lo stato a `completed` e registra eventuali note di esecuzione ([Appendice C]). Gli artefatti prodotti dalla *pipeline* vengono salvati nella cartella dedicata `outputs/{processId}/` (mappe, tabelle, log e report; cfr. § 2.5.1). Nella distribuzione Docker, tali output possono essere persistiti sul filesystem host tramite volumi, in modo da rendere i risultati riutilizzabili e ispezionabili anche dopo il riavvio del container. Eventuali errori provenienti dagli script Python vengono intercettati dal servizio di orchestrazione, mappati su `status = "error"` e riportati nei log locali del relativo processo.

La pipeline Python (script *headless*) esegue l'analisi iniziale del DEM e produce le derivate e i file di output, scrivendoli in `outputs/{processId}/` con i relativi log operativi; al passaggio dello stato a `completed`, l'interfaccia abilita gli step successivi (selezione del modello e pannello risultati) e collega direttamente ai prodotti generati ([Appendice A], § 2.5.1).

Questa ripartizione delle responsabilità mantiene l'architettura leggera e di fatto *stateless*: la persistenza di processo è affidata all'identificativo `processId` e alla serializzazione su disco, in linea con le buone pratiche dei servizi REST che evitano dipendenze da database o sessioni permanenti [Castronova et al., 2013].

La Figura 3 mostra l'interfaccia di caricamento e validazione del DEM (*Upload panel*).

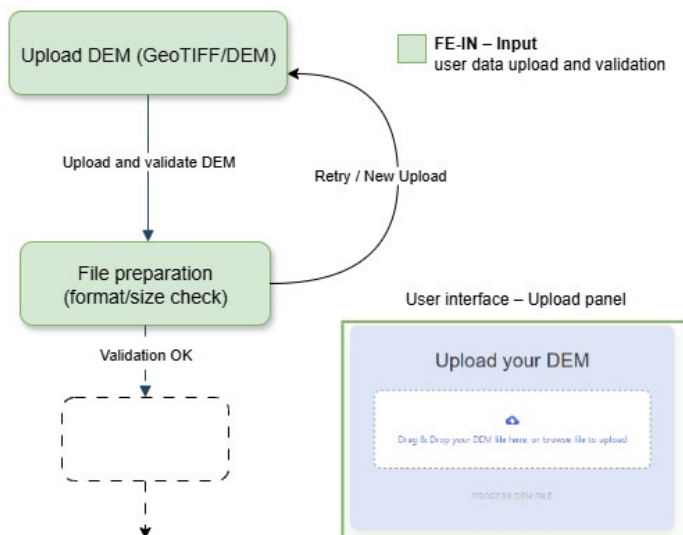


Figura 3 Diagramma di flusso della fase di caricamento e validazione del DEM (FE-IN), con vista dell'Upload panel.

Figure 3 Flow diagram of the DEM upload and validation stage (FE-IN), including the Upload panel view.

2.2 Derivative Computation (Python $\rightleftharpoons$ Node; triggered from FE)

Il blocco Derivative Computation è il primo stadio di elaborazione morfometrica automatizzata, attivato dal *frontend* a seguito della validazione del DEM. Quando l'utente preme "PROCESS DEM FILE", il servizio Node/Express riceve il file e i metadati, genera un identificativo univoco

di processo (processId) e avvia in modalità *headless* lo script Python `complete_dem_analysis.py`, responsabile del calcolo delle derivate topografiche fondamentali ([Appendice C]). Durante l'esecuzione, il *frontend* mantiene la sincronizzazione con lo stato del *job* tramite richieste periodiche `GET /processStatus/{processId}`, aggiornando la barra di avanzamento e la messaggistica attraverso l'*hook* React `useProcessStatus` ([Appendice A]). Al completamento, Node/Express riceve la *callback* `POST /processComplete/{processId}`, imposta lo stato a `completed` e rende disponibili gli artefatti nel percorso `outputs/{processId}/` (§ 2.5.1).

La Figura 4 illustra il flusso completo della fase di Derivative Computation, mostrando la chiamata iniziale dello script Python `complete_dem_analysis`, le principali derivate morfometriche calcolate e le opzioni di esportazione dei risultati nel *frontend*.

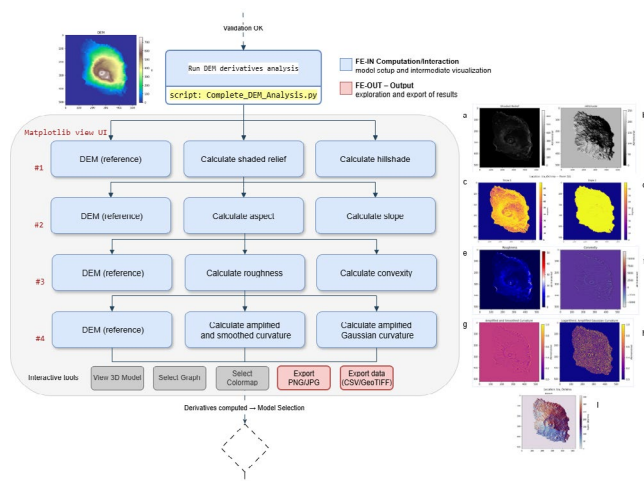


Figura 4 Schema del blocco “Derivative Computation”, con generazione e visualizzazione delle derivate topografiche (a-b: shaded relief-ombreggiatura; c-d: esposizione-pendenza; e-f: rugosità-curvatura amplificata; g-h: curvatura amplificata e levigata-curvatura gaussiana).

Figure 4 Schematic view of the “Derivative Computation” block, showing the generation and visualization of topographic derivatives (a–b: shaded relief-hillshade; c–d: aspect-slope; e–f: roughness-amplified curvature; g–h: amplified & smoothed curvature-log-amplified Gaussian curvature).

2.2.1 Script Structure and Computational Workflow

In questa fase la *pipeline* esegue `complete_dem_analysis.py`, che costituisce il nucleo computazionale del sistema per la produzione delle derivate. Lo script legge la griglia DEM tramite rasterio e calcola le grandezze morfometriche mediante operatori NumPy/SciPy ottimizzati per l'elaborazione matriciale; nel *job* /process le derivate sono riassunte in pannelli compositi e panoramiche salvati su disco (`dem_overview.png`, `triplet_*`, `double_*`) ed elencati in `manifest.json`. I PNG per-derivata (es. `shaded_relief.png`, `slope.png`, `curvature.png`) sono generati su richiesta tramite gli endpoint dedicati (cfr. § 2.6).

Le funzioni restituiscono array NumPy con valori in gradi (pendenza) o adimensionali (curvature, rugosità) e registrano tempi/picchi RAM con prefissi di log [TIMING] e [MEMORY] per consentire valutazioni prestazionali e scenari di scalabilità.

Le derivate calcolate dal modulo sono:

- Ombreggiatura (Hillshade), prodotto di visualizzazione ottenuto da pendenza e aspetto con illuminazione sintetica (parametri di default: azimuth 45° e angolo di elevazione della sorgente 45°; modificabili). Nota: l'hillshade con sorgente luminosa direzionale può introdurre azimuth bias nella percezione di alcune morfologie [Favalli e Fornaciai, 2017];

per questo è affiancato da derivate oggettive (es. pendenza, curvatura, rugosità) per l'analisi e il controllo qualità. *Funzione:* `calculate_hillshade`.

- Shaded relief, composito di resa del rilievo salvato, quando richiesto, come `shaded_relief.png`.
- Esposizione (Aspect), direzione della massima pendenza in gradi (0–360°), ricavata dalle componenti del gradiente. *Funzione:* `calculate_aspect`.
- Pendenza (Slope / Slope_2), modulo del gradiente espresso in gradi, in due varianti equivalenti per confronto di robustezza; la seconda utilizza Sobel, impiegato anche nelle *pipeline* volumetriche. *Funzioni:* `calculate_slope`, `calculate_slope_2`.
- Curvatura amplificata (Convexity), Derivata dalla Laplaciana ($\partial^2 z / \partial x^2 + \partial^2 z / \partial y^2$), amplificata mediante un fattore costante ($\times 200$) per esaltare rapidamente forme concave e convesse anche deboli. *Funzione:* `calculate_convexity`.
- Curvatura amplificata e levigata (Amplified Gaussian Curvature) derivata dalla Laplaciana “base”, amplificata ($\times 5$) e successivamente filtrata mediante smoothing gaussiano ($\sigma = 1$), quindi normalizzata su [0–1] per stabilizzarne la resa cartografica.
- Pipeline: `calculate_curvature` → amplificazione → `gaussian_filter` → normalizzazione.
- Curvatura gaussiana log-amplificata (Log-Amplified Gaussian Curvature), calcolata come prodotto delle curvature principali, con smoothing gaussiano ($\sigma = 1$), amplificazione ($\times 10$), soglia sui valori negativi e scalatura logaritmica ($\log_{10}$) per aumentarne il contrasto e ridurre l'effetto delle code dei valori estremi; segue normalizzazione su [0–1]. *Funzione:* `calculate_gaussian_curvature`.
- Rugosità (Roughness), deviazione standard locale su finestra mobile 3×3, indicativa delle micro-irregolarità topografiche e più costosa computazionalmente. *Funzione:* `calculate_roughness`.

Il flusso risultante, illustrato in Figura 4, parte dalla chiamata a `complete_dem_analysis` e produce progressivamente le derivate con anteprime Matplotlib visualizzabili in interfaccia e opzioni di *export* in PNG/JPG e CSV/GeoTIFF. Il passaggio dello stato a `completed` consente la transizione al blocco successivo di Model Selection.

L'esecuzione delle derivate è a carico della *pipeline* Python, mentre il servizio Node/Express ne governa l'avvio e lo stato esposto al *frontend*, che resta responsabile del solo innesco e del *polling* informativo.

2.2.2 Headless execution and performance environment

L'esecuzione avviene integralmente in modalità headless, per compatibilità con ambienti server/container e per evitare dipendenze da interfacce grafiche. Nella release v0.1.1 (Docker edition) l'ambiente runtime e le librerie sono incluse nell'immagine: l'utente finale interagisce via browser e non deve installare dipendenze via pip sul sistema host. Per esecuzioni fuori Docker (sviluppo o debug), l'ambiente può essere ricreato in venv/Conda installando le dipendenze da `requirements.txt` (es. `pip install -r requirements.txt`).

Impostazioni operative raccomandate includono:

- `MPLBACKEND=Agg`
- `QT_QPA_PLATFORM=offscreen`
- `OPENBLAS_NUM_THREADS=1` (rendering off-screen e comportamento deterministico in single-thread)

L'interfaccia Node/Express rimane agnostica rispetto alla GUI: invia parametri, riceve stdout/stderr e organizza gli artefatti in `outputs/{processId}/`.

Un'implementazione rappresentativa è Tethys WPS Server, che mostra come l'esposizione dei processi WPS favorisca interoperabilità e riuso tra modelli eterogenei [Qiao et al., 2019]; nel presente progetto l'esposizione è REST locale.

2.2.3 Schema sintetico degli algoritmi (DEM → contorni → volumi)

Di seguito si riporta uno schema di alto livello (non esaustivo) dei passaggi algoritmici; i dettagli implementativi sono nelle sezioni richiamate.

Input (DEM) e requisiti minimi. MorphoVolc assume in input un DEM GeoTIFF georiferito (CRS/EPGS valido), con valori di quota in unità metriche (tipicamente metri) e assenza di estese aree NoData nell'area dell'edificio. L'analisi è pensata per edifici a rilievo positivo e morfologie relativamente regolari; in presenza di strutture fissurali/maar o topografie molto complesse, la qualità dei contorni e delle stime può degradare (§5.2).

Derivate morfometriche. A partire dal DEM vengono calcolati i principali layer di supporto (hillshade/shaded relief, slope, curvature nelle varianti implementate, roughness) impiegati sia per la lettura/QA in interfaccia sia come ausilio alla selezione dei contorni (§2.2).

Estrazione dei contorni (base e caldera). I contorni candidati sono derivati da soglie operative sull'elevazione normalizzata (default: base ~5% e caldera ~80% di z_{max} nella versione v0.1.1), con estrazione di isolinee mediante funzioni standard di contouring e selezione del candidato principale in base a criteri geometrici (continuità, dimensione/area, coerenza geometrica e topologica). Per la caldera, oltre alla soglia altimetrica, viene usata un'informazione di gradiente (mappa Sobel) per identificare un bordo interno plausibile e i relativi punti "opposti" (§2.3.2).

Punti opposti e misure geometriche. Sul contorno di base vengono individuate coppie di punti antipodali per stimare larghezze/assi principali; sulla caldera i punti opposti sono ricavati attorno al massimo gradiente e al relativo antipodale. Le aree dei contorni sono calcolate con formula di Gauss (shoelace) e le distanze con metrica euclidea (§2.4.1).

Stima volumetrica (modelli analitici). Il volume dell'edificio è stimato con un modello di tronco di cono (circolare o ellittico), usando le aree/assi della base e una misura d'altezza definita rispetto a z_{max} (v0.1.1- scelta operativa della release; vedi roadmap per alternative). Quando presente, la caldera è modellata come cavità con due approssimazioni alternative: (i) semisfera/semiellissoide (Approx1) oppure (ii) cilindro/cilindro ellittico (Approx2). Il volume "effettivo" è calcolato come $V_{eff} = V_{edificio} - V_{caldera}$ (§2.3.2; Figura 6).

Output e tracciabilità. Ogni esecuzione produce un pacchetto di artefatti (GeoJSON/CSV/PNG/PDF) organizzato per processId, con manifest e log [TIMING]/[MEMORY] per riproducibilità e QA (§2.5; Tabella 1).

La gestione dell'ambiente headless e della serializzazione degli artefatti è demandata alla pipeline Python, con Node/Express in ruolo di orchestratore stateless e il frontend limitato alla supervisione dell'avanzamento. La sequenza è riassunta graficamente in Figura 1.

2.3 Model Selection and Volumetric Computation (React $\rightleftharpoons$ Node $\rightleftharpoons$ Python)

Il blocco Model Selection rappresenta la fase intermedia del flusso operativo di *MorphoVolc DEM Interface*, in cui l'utente, una volta completato il calcolo delle derivate topografiche (§ 2.2), seleziona il modello volumetrico più adatto alla morfologia del vulcano analizzato. Questa scelta guida l'attivazione di uno dei quattro script Python dedicati al calcolo volumetrico, che implementano le due geometrie principali (circolare ed ellittica) con le rispettive approssimazioni.

La logica di questa fase è duplice: da un lato l'interfaccia React fornisce all'utente un modulo di selezione chiaro e parametrico; dall'altro, il servizio Node/Express orchestra la chiamata al backend Python, trasmettendo le opzioni di modello e gestendo la creazione e la chiusura del processo di calcolo.

La Figura 5 illustra il flusso generale della fase di *Model Selection*, dal momento della scelta nel *frontend* fino all'invocazione degli script Python e alla generazione dei primi risultati volumetrici intermedi.

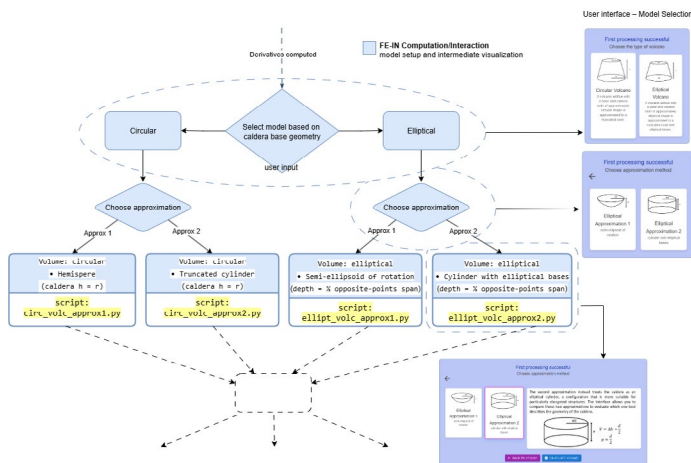


Figura 5 Scelta del modello volumetrico (circolare/ellittico) e dell'approssimazione (1/2).

Figure 5 Volumetric model selection (circular/elliptical) and approximation type (1/2).

2.3.1 Model Selection Logic and User Interaction

Dal punto di vista funzionale, il modulo `VolumeSelection` dell'interfaccia React (file `VolumeSelection.js`) costituisce il cuore di questa fase e presenta all'utente un pannello interattivo con due gruppi di opzioni:

- tipo di geometria — *circular* o *elliptical*;
- approssimazione volumetrica — 1 (semisfera o semiellissoide) oppure 2 (cilindro o cilindro ellittico).

A scelta avvenuta, il componente genera una richiesta POST `/calculateVolume` verso il servizio Node/Express (esposto localmente dal container), trasmettendo:

- il file DEM o, se già processato, il riferimento `processId`;
- i parametri `volumeType` e `approximationType`;
- eventuali metadati del processo (durata lato client, fase lato backend).

Il *frontend* gestisce la visualizzazione dinamica dello stato di esecuzione, misurando il *runtime* totale lato client e intercettando l'header di risposta `x-server-phase`, che riporta la fase di calcolo corrente. In parallelo, il servizio Node riceve la richiesta, identifica la combinazione selezionata e avvia lo script Python corrispondente tra i quattro disponibili:

- `CircularVolcano_Approx1.py`
- `CircularVolcano_Approx2.py`
- `EllipticalVolcano_Approx1.py`
- `EllipticalVolcano_Approx2.py`

Se viene fornito un `processId`, l'elaborazione riusa gli artefatti già prodotti nella fase §2.2; in caso contrario viene inizializzato un nuovo processo.

Il *frontend* aggiorna progressivamente lo stato del *job* attraverso le chiamate GET /processStatus/{processId} e abilita la visualizzazione del pannello *Results* al termine dell'elaborazione ([Appendice A]).

Questa architettura assicura un flusso asincrono controllato: l'utente mantiene la piena percezione delle fasi di calcolo, mentre il servizio Node si occupa esclusivamente dell'orchestrazione e della serializzazione degli artefatti e dei metadati essenziali su filesystem (outputs/{processId}/...) ([Appendice D], § 2.5.1).

2.3.2 Volumetric computation workflow (Python scripts)

La computazione volumetrica vera e propria è eseguita dalla *pipeline* Python, che riceve in input il DEM già elaborato e le informazioni di modello trasmesse dal servizio Node.

Le principali dipendenze software sono:

- Python: numpy ≥ 1.21 e scikit-image ≥ 0.18 (estrazione dei contorni con `measure.find_contours`, gradienti/Sobel), reportlab ≥ 3.6 per il PDF lato backend (ReportLab) distinto dal PDF rapido lato *frontend* (jsPDF).
- Node/Express: express $\geq 4.21.1$, multer $\geq 1.4.5$ -lts.1, cors $\geq 2.8.5$, uuid $\geq 11.0.3$.
- *Frontend*: axios $\geq 1.7.x$ per la chiamata POST /calculateVolume.

Gli script CircularVolcano_Approx1/2.py e EllipticalVolcano_Approx1/2.py implementano le quattro combinazioni geometriche previste:

- geometria circolare: edificio a tronco di cono e caldera modellata come semisfera (Approx 1) o cilindro (Approx 2);
- geometria ellittica: edificio a tronco di cono ellittico e caldera approssimata da semiellissoide di rotazione (Approx 1) o cilindro a basi ellittiche (Approx 2).
- Ogni script applica lo stesso schema generale:
 - lettura del DEM e dei derivati topografici necessari all'individuazione dei contorni;
 - estrazione e selezione automatica dei contorni principali (base e caldera) tramite `skimage.measure.find_contours`;
 - identificazione dei punti opposti — per la base tramite punti antipodali, per la caldera tramite il massimo del gradiente (mappa Sobel) e relativo antipodale;
 - calcolo delle aree e delle distanze con metodi analitici (formula di Gauss (shoelace), distanza euclidea).

Applicazione delle formule volumetriche analitiche definite per il modello selezionato. Le Figure 6a–b riassumono le geometrie e le formulazioni adottate per i casi circolare (a) ed ellittico (b), con edificio modellato come tronco di cono e due differenti approssimazioni di caldera (semisfera/semiellissoide e cilindro/cilindro ellittico).

Il risultato di ciascuna esecuzione consiste in un set di parametri e valori numerici scritti in `metrics.csv`, accompagnato dai contorni vettoriali (`contours_base.geojson`, `contours_caldera.geojson`) e da un report opzionale in formato PDF. Questi artefatti costituiscono la base per la successiva fase di Volume Calculation (§ 2.4), dove verranno integrati con le statistiche morfometriche e presentati all'utente nel pannello finale dei risultati.

Dal punto di vista architetturale:

- il *frontend* React gestisce la selezione e l'avvio;
- il servizio Node/Express funge da orchestratore, assicurando la sequenza corretta delle chiamate e la gestione degli artefatti nel filesystem;
- la *pipeline* Python realizza i calcoli numerici e produce i risultati persistenti per ogni processId.

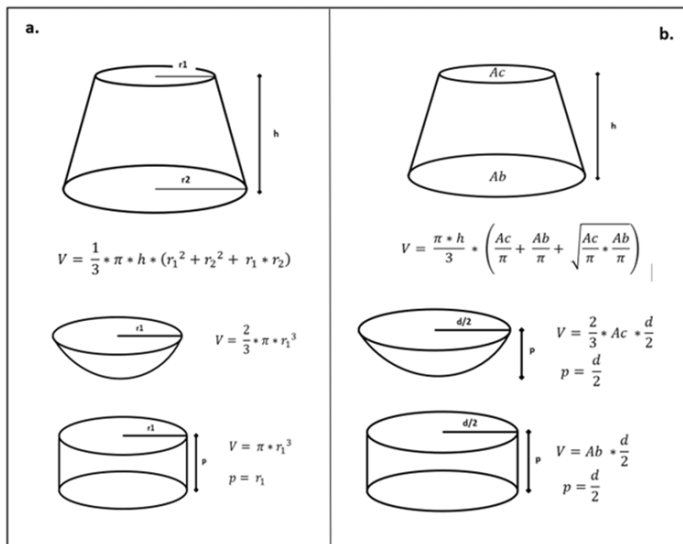


Figura 6 Forme e formule volumetriche per le geometrie circolare (a) ed ellittica (b).

Figure 6 Volumetric shapes and formulas for circular (a) and elliptical (b) geometries.

2.4 Volume Calculation & Results Preparation (FE-IN $\rightleftharpoons$ FE-OUT)

Il blocco Volume Calculation rappresenta la fase di chiusura del calcolo volumetrico e di preparazione ai risultati finali.

In questo stadio, i prodotti numerici generati dagli script Python vengono aggregati e validati, costituendo la base per la successiva fase di Results & Export (§ 2.5).

L'obiettivo è consolidare i valori volumetrici e le geometrie derivate, rendendoli pronti per la visualizzazione e l'esportazione nell'interfaccia utente. Questa fase consolida le metriche volumetriche prodotte dagli script Python con le informazioni spaziali e grafiche, preparando la visualizzazione nell'interfaccia.

Quando il servizio Node riceve la conferma di completamento della computazione (*callback* POST /processComplete/{processId}), raccoglie i file generati nella directory outputs/{processId}/ e li rende accessibili alla vista di ispezione del *frontend*, propedeutica al modulo di Results & Export (§ 2.5).

I file prodotti includono:

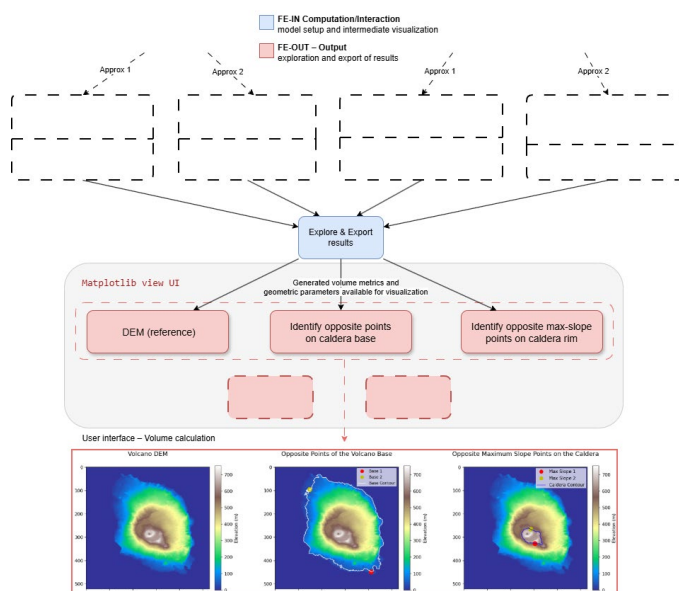
- metrics.csv (con parametri geometrici e volumetrici: V_total, V_caldera, V_eff, A_base, A_caldera, r1, r2, h, p);
- contours_base.geojson e contours_caldera.geojson (perimetri principali);
- un report opzionale report.pdf con le tabelle di sintesi e le miniature grafiche.

Inoltre, dalla fase /process risultano disponibili le panoramiche/pannelli salvati come dem_overview.png, triplet_* e double_*, insieme al manifest.json di esecuzione. I PNG per-derivata (es. shaded_relief.png, slope.png) sono presenti solo se invocati gli endpoint specifici (§ 2.6).

La Figura 7 illustra il flusso operativo di questa fase, dal momento della ricezione dei file al rendering dei risultati intermedi (pre-output).

Figura 7 Flusso di calcolo dei volumi e individuazione dei punti opposti su base e caldera.

Figure 7 Volume computation workflow and identification of opposite points on base and caldera.



2.4.1 Computation and Data Aggregation

La computazione finale e l'aggregazione dei dati sono eseguite in cooperazione tra la *pipeline* Python e il servizio Node/Express.

La *pipeline* elabora i risultati numerici e geometrici provenienti dagli script di approssimazione, mentre Node si occupa di raccogliarli, serializzarli e renderli disponibili in forma strutturata per il *frontend*.

Le principali funzioni impiegate nella fase di calcolo e aggregazione sono:

- `find_opposite_base_points()` – individua i punti antipodali della base vulcanica;
- `find_opposite_slope_points()` – identifica i punti di massimo gradiente nella caldera e i relativi opposti;
- `calculate_area()` – calcola le aree planimetriche dei contorni principali mediante la *shoelace formula*;
- `distance_between_points()` – stima le distanze lineari tra coppie di punti, utilizzate come semiassi o raggi nelle formule volumetriche.

Le informazioni così ottenute vengono raccolte in `metrics.csv`, che rappresenta l'output numerico principale per ogni processId.

In parallelo, i contorni vettoriali sono esportati come GeoJSON (`contours_base.geojson`, `contours_caldera.geojson`) e resi interoperabili con software GIS (Geographical Information System) esterni.

Il servizio Node integra queste informazioni e aggiorna lo stato del processo, marcandolo come completed una volta verificata la presenza di tutti gli artefatti previsti.

2.4.2 Visualization and Export (React UI $\rightleftharpoons$ Node)

L'interfaccia fornisce una vista di esplorazione dei risultati volumetrici (pre-output), utile per la verifica visiva e il controllo qualità prima della fase di Results & Export.

Le principali dipendenze sono:

- Python: `matplotlib` $\geq$ 3.4 (render figure/anteprime), `reportlab` $\geq$ 3.6 (report PDF di anteprima).
- Node/Express: modulo `fs` per lettura file su disco; eventuale `express.static` per la servitizzazione degli artefatti in `outputs/{processId}/`.

- *Frontend* (React): `axios` $\geq 1.7.x$ per GET `/processStatus/{processId}` e download (opzionale per *export* locale).

La vista è composta da tre pannelli principali:

- DEM di riferimento, utilizzato come base per la contestualizzazione topografica;
- contorni della base e della caldera, sovrapposti e colorati in modo differenziato;
- punti opposti, evidenziati con marker distintivi per la verifica visiva delle approssimazioni geometriche.

In questa fase, in forma intermedia, l'interfaccia consente l'*export* per verifica/QA (Quality Assurance) tramite pulsanti dedicati:

- immagini (PNG/JPG) delle viste di controllo;
- raster GeoTIFF e tabelle CSV dei dati quantitativi essenziali;
- report di anteprima (PDF) generato con `ReportLab`, contenente miniature e parametri chiave.

Le opzioni di impaginazione avanzata e il pacchetto di *export* completo sono trattati nella fase successiva. Durante la fase di visualizzazione, il *frontend* utilizza `axios` per interrogare il servizio Node sullo stato del processo (GET `/processStatus/{processId}`) e per scaricare i file aggiornati. Node, a sua volta, legge i file già presenti su disco e li restituisce senza ricalcolo, mantenendo la logica *stateless* e la separazione tra calcolo e presentazione.

Il completamento della sessione viene confermato con POST `/processComplete/{processId}` (callback server-side invocata al termine della pipeline, non un'azione manuale dell'utente), che marca definitivamente lo stato come *completed* e conclude la fase di calcolo, abilitando i passaggi di Results & Export per il relativo `processId`.

Questa organizzazione consente di mantenere la *pipeline* modulare, riproducibile coerente con i principi REST e con pattern tipici di servizi geoprocessing (WPS-like) garantendo la tracciabilità completa del processo e la possibilità di rieseguire analisi successive sugli stessi artefatti (§ 2.5.1).

2.5 Results & Export (FE-OUT)

Il blocco Results & Export conclude il flusso operativo rendendo disponibili, in forma consultabile via web UI e scaricabile tutti gli artefatti prodotti dal processo: raster/immagini di supporto, contorni vettoriali, tabelle numeriche e report PDF. L'obiettivo è fornire un pacchetto tracciabile per ogni `processId`, pronto per analisi successive o archiviazione.

Gli artefatti sono organizzati per esecuzione nella cartella `outputs/{processId}/`, tipicamente includendo: panoramiche e pannelli della fase `/process` (`dem_overview.png`, `triplet_*`, `double_*`, `manifest.json`), i contorni principali (GeoJSON), la tabella delle metriche (`metrics.csv`) e, ove richiesto, il report (`report.pdf`). I derivati singoli (es. `shaded_relief.png`, `slope.png`) sono disponibili se generati on-demand tramite gli endpoint dedicati (§ 2.6). Questa convenzione garantisce una provenienza chiara e integrazione agevole in flussi *batch*.

Le principali dipendenze sono:

- Python (backend report): `reportlab` ≥ 3.6 (composizione PDF finale); se generi miniature lato backend: `matplotlib` ≥ 3.4 .
- Node/Express (serving file): `express` $\geq 4.21.1$ (route download), modulo `fs` (lettura da `outputs/{processId}/`), `cors` $\geq 2.8.5$.

Frontend (FE-OUT): `axios` $\geq 1.7.x$ (download), opzionale `jspdf` $\geq 2.5.x$ (PDF "quick" lato client). La Figura 8 illustra le modalità di esportazione dei risultati e la generazione del report PDF all'interno dell'interfaccia.

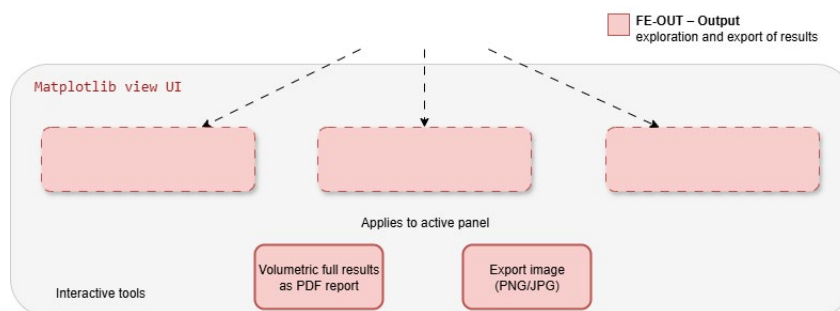


Figura 8 Moduli di esportazione e generazione del report PDF finale.

Figure 8 Export modules and final PDF report generation.

2.5.1 Exportables & File Layout

In interfaccia, una vista FE-OUT offre i pulsanti di download per:

- immagini: anteprime e figure (PNG/JPG) per la documentazione rapida e la verifica visiva;
- raster e tabelle: GeoTIFF dei prodotti derivati e CSV con parametri geometrici/volumetrici (es. V_{total} , $V_{caldera}$, V_{eff} , A_{base} , $A_{caldera}$, $r1$, $r2$, h , p) estratti da `metrics.csv`;
- report PDF: documento sintetico con intestazione, tabella *Description/Value* e miniature delle viste.

Dal lato *frontend*, un componente dedicato consente anche la generazione locale di un PDF “rapido” tramite jsPDF, con header grafico (immagine di sfondo + logo), titolo, linea separatrice e una tabella a due colonne popolata tramite parsing dinamico una stringa di risultati; il salvataggio avviene come `CalculationResults.pdf`. Questo meccanismo è pensato per esportazioni istantanee dal browser.

2.5.2 Report Generation

- *Frontend* (React) – gestisce i pulsanti di download, compone eventuali PDF “quick” con jsPDF (header, logo, tabella con parsing etichetta/valore tramite regex), e invoca gli endpoint del servizio per recuperare i file aggiornati associati al `processId`.
- Servizio Node/Express – funge da orchestratore *stateless*: espone gli URL di download, legge i file già presenti in `outputs/{processId}/` e li ritorna senza ricalcolo, mantenendo separati calcolo e presentazione. L'assenza di DB e l'organizzazione per cartella di processo sono coerenti con l'architettura complessiva.
- *Pipeline* Python (ReportLab) – per il report completo lato backend utilizza ReportLab: compone un header con logo + immagine di testata, applica il titolo, quindi costruisce una tabella “Description/Value” iterando su `results_list`. Lo stile della tabella (header grigio, grid, allineamenti) e l'inclusione di miniature vengono gestiti nel generatore; i file di testata (logo/background) sono letti dal filesystem e verificati prima della composizione.

2.5.3 Download Workflow

Nella vista FE-OUT l'utente accede a una griglia di pannelli che combina il DEM di riferimento con la sovrapposizione dei contorni di base e di caldera e, quando pertinenti, con i marker

dei punti opposti sul bordo. Da questa schermata è possibile effettuare la validazione visiva dei risultati e procedere al download degli artefatti associati al relativo processId. I formati disponibili comprendono immagini PNG/JPG, raster GeoTIFF, tabelle CSV e il PDF di report; l'organizzazione per cartella garantisce coerenza con la struttura degli output e facilita il recupero selettivo dei file. Nel complesso, il bundle per-processo (GeoJSON/CSV/PNG/JPG/PDF) risulta portabile e tracciabile, con file riproducibili e riutilizzabili in linea con le buone pratiche operative già richiamate.

2.6 Application Routes and Artifacts Table

Endpoint	Input (form/body)	Effetto lato servizio	Output (HTTP)	Artefatti su disco
POST/process	demFile (file), originalFileName (opz.)	Crea processId, avvia complete_ dem_analysis.py (detached), stato processing	{ "processId": "<uuid>" }	outputs/{processId}/ dem_overview. png; triplet_*. png; double_*.png; manifest.json
GET/processStatus/ {processId}	–	Legge lo stato in memoria	{ "status": "processing" "completed" "error" }	–
POST/ processComplete/ {processId}	(callback Python)	Marca stato completed e registra note	{ "ok": true }	–
POST/ calculateVolume	demFile (file) oppure processId; volumeType {circular,elliptical}; approximationType {1,2}	Avvia script volumetrico, raccolge stdout/ stderr, (opz.) genera PDF	{ "result": "<string>", "metricsPath": "outputs/<processId>/ metrics.csv" }	outputs/{processId}/ metrics.csv; contours_base. geojson; contours_ caldera.geojson; (opz.) report.pdf
POST/shadedRelief	demFile (file) oppure processId	Genera shaded relief (derivato dedicato)	{ "result": "Shaded relief generated" }	outputs/{processId}/ shaded_relief.png
POST/ calculateSlopes	demFile (file) oppure processId	Calcola slope e slope_2 (derivati dedicati)	{ "result": "Slopes calculated" }	outputs/{processId}/ slope.png; slope_2. png
POST/ calculateCurvatures	demFile (file) oppure processId	Calcola curvature (laplaciana + gaussiana)	{ "result": "Curvatures calculated" }	outputs/{processId}/ curvature.png; gaussian_curvature. png

Tabella 1 Endpoint REST del servizio Node/Express: input accettati, effetti lato server, output HTTP e artefatti su filesystem per ogni processId.

Table 1 Node/Express REST endpoints: accepted inputs, server-side effects, HTTP outputs, and filesystem artifacts for each processId.

2.7 Dependencies and Environment

2.7.1 Runtime version

Nella release v0.1.1 (Docker edition) lo stack è eseguito all'interno di un'immagine basata su Node.js 20 (Debian bookworm) con un ambiente Python isolato in virtualenv. Le versioni effettive di Node/npm e del runtime Python sono quelle incluse nell'immagine di release e sono tracciabili nel Dockerfile e nei log di build.

Le analisi morfometriche e la generazione dei prodotti avvengono in modalità file-based: tutti gli artefatti sono scritti e letti dal filesystem in `outputs/{processId}/` senza uso di database.

2.7.2 Frontend (React) — package.json highlights

Le dipendenze del frontend sono definite nel file `frontend/package.json` e includono la libreria React e i principali componenti UI; nella Docker release vengono installate e buildate automaticamente durante la fase di build dell'immagine.

2.7.3 Backend (Node/Express) — package.json highlights

Servizio di orchestrazione e serving degli artefatti:

- `express ^4.21.1`, `cors ^2.8.5`, `multer ^1.4.5-lts.1`, `uuid ^11.0.3`. (Nessuna libreria di data processing lato Node: il parsing numerico resta lato Python o via file CSV/JSON.).

Script di esecuzione: `node server.js (start)`.

2.7.4 Python pipeline — dependencies

La pipeline Python esegue il calcolo delle derivate, la selezione dei contorni e le computazioni volumetriche; inoltre compone il report (PDF) lato backend quando richiesto. Le librerie utilizzate includono:

- Numerics & imaging: `numpy`, `scikit-image`, `matplotlib`.
- Raster I/O: `rasterio` (stack GDAL/PROJ in container).
- Report PDF: `reportlab`.
- Runtime helpers: `psutil`, `requests`, `pandas` (telemetria/logging e gestione output strutturati).
- Opzionali (solo se si espone la pipeline Python come microservizio): `fastapi`, `uvicorn`[standard].

Headless settings. Per l'esecuzione in ambienti server e containerizzati: `MPLBACKEND=Agg`, `QT_QPA_PLATFORM=offscreen`, `OPENBLAS_NUM_THREADS=1` (§ 2.2.2).

Nota operativa (installazioni locali fuori Docker, sviluppo/debug): su Windows pacchetti come `rasterio` dipendono da GDAL/PROJ; in questi casi si consiglia l'uso di Conda/conda-forge o di wheel precompilate per la versione di Python in uso.

2.7.5 Deployment and environment notes

- Organizzazione artefatti: `outputs/{processId}/` contiene panoramiche/pannelli generati in `/process` (`dem_overview.png`, `triplet_*`, `double_*`, `manifest.json`), i contorni (`contours_*.geojson`), le metriche (`metrics.csv`) e, se richiesto, `report.pdf`.
- Persistenza in Docker: `uploads` e `outputs` sono mappati su directory locali tramite bind mounts (`docker-compose.yml`), preservando gli artefatti anche dopo il riavvio del container.
- Stateless orchestration: Node/Express espone endpoint locali e serve i file senza ricalcolo, mantenendo separati calcolo e presentazione.
- Build FE/BE: il frontend è basato su Create React App (CRA); il backend è un servizio Express minimal.
- Variabili d'ambiente: includere in un `.env/.env.example` le sole chiavi non sensibili (vedi `headless settings` sopra).

Per una rassegna sintetica dei problemi più comuni durante installazione ed esecuzione (es. mancato riconoscimento del CRS, errori di memoria, incompletezza degli artefatti), si veda Appendice E – Troubleshooting rapido.

3. Casi d'uso e prestazioni

3.1 Esempi applicativi (Miyakejima e Izu-Ōshima)

Per la validazione funzionale e la dimostrazione delle capacità operative di MorphoVolc DEM Interface sono stati selezionati due edifici vulcanici giapponesi – Miyakejima e Izu-Ōshima – appartenenti all'arco vulcanico Izu-Bonin. La scelta è motivata dalla disponibilità di DEM a scala regionale e da una morfologia sufficientemente regolare da consentire il confronto diretto fra modelli volumetrici circolari ed ellittici.

Il flusso di lavoro ha seguito lo schema standard, utilizzando DEM SRTM (Shuttle Radar Topography Mission) e ASTER GDEM (Advanced Spaceborne Thermal Emission and Reflection Radiometer – Global DEM) entrambi a 30 m come *dataset* di riferimento.

Per ciascun caso, la *pipeline* ha prodotto i seguenti artefatti:

- Panoramiche/pannelli della fase `/process` (`dem_overview.png`, `triplet_*`, `double_*`) e, se richiesti tramite endpoint dedicati, i derivati singoli (es. `shaded_relief.png`, `slope.png`, `curvature.png`).
- Contorni base e caldera in formato GeoJSON.
- Tabelle metriche (aree, assi, volumi) in formato CSV.
- Report PDF riepilogativo con miniature delle viste e tabella Description/Value.

Per Miyakejima, il modello ellittico ha evidenziato una migliore aderenza alla topografia reale, con un volume netto inferiore del 7 % rispetto al modello circolare, dovuto alla maggiore eccentricità della base. Per Izu-Ōshima, invece, le due approssimazioni restituiscono valori coerenti entro il 3 %, confermandone la robustezza nelle applicazioni operative e i valori sono ottenuti con lo stesso DEM (30 m) e identiche soglie operative; il `processId` dell'esecuzione è riportato nel report PDF (se generato) ed è sempre indicato in `manifest.json` nella cartella `outputs/{processId}/` (vedi § 2.5.1).

Gli artefatti generati risultano completamente tracciabili: ogni esecuzione produce una cartella dedicata `outputs/{processId}/` contenente i file raster, vettoriali e numerici, oltre al log temporale e di memoria. La riproducibilità è supportata da un `manifest.json` che elenca gli artefatti prodotti e i timestamp di esecuzione (eventuali parametri/versioni, se configurati).

3.2 Minibenchmark e metodologia di misura

Un minibenchmark è stato condotto sui due casi studio per valutare tempi e memoria dei moduli principali; i risultati sintetizzati in Tabella 2 mostrano che il costo è dominato dalle operazioni raster (curvatura/ rugosità, ~65 % del *runtime*). La *pipeline* supporta *batch* tramite riuso del DEM (processId persistente) e mantiene picchi < 700 MB fino a ~1200×1200 px, compatibili con ambienti a risorse limitate. I tempi end-to-end includono I/O e latenza di servizio (REST/WPS), un fattore rilevante quando i payload sono frequenti e piccoli. Le performance sono coerenti con librerie ottimizzate (NumPy, rasterio, scikit-image) e possono migliorare con parallelizzazione e caching dei *layer* intermedi. La tracciabilità è garantita da log e manifest di run. I tempi e i picchi di memoria sono ricavati dai log con prefissi [TIMING] e [MEMORY] emessi dal servizio.

Caso studio	Risoluzione DEM	Runtime totale (s)	Memoria di picco (MB)
Miyakejima	30 m (SRTM)	48	612
Izu-Ōshima	30 m (ASTER GDEM)	52	645

Tabella 2 Minibenchmark dei casi studio: runtime totale e memoria di picco. Misure effettuate su workstation Linux (Ubuntu 22.04, Intel® Core™ i7-11800H @ 2.3 GHz, 32 GB RAM), ambiente Conda, backend grafico headless (MPLBACKEND=Agg). Valori ricavati dai log del servizio Node/Express.

Table 2 Mini-benchmark for the case studies: total runtime and peak memory. Measurements on a Linux workstation (Ubuntu 22.04, Intel® Core™ i7-11800H @ 2.3 GHz, 32 GB RAM), Conda environment, headless graphics backend (MPLBACKEND=Agg). Values taken from Node/Express service logs.

3.3 Confronto metodologico

Per confronto metodologico, NETVOLC su 56 coni del Mauna Kea mostra accordo con contorni manuali in ~2/3 dei casi; dove necessario, l'uso di funzioni di costo alternative migliora i risultati. Le differenze su larghezza/altezza sono per lo più < 10 % e il rapporto H/W varia entro ~6 % [Euillades et al., 2013]. Questi risultati confermano l'utilità delle *pipeline* automatiche per ottenere valutazioni morfometriche di riferimento rapide e riproducibili.

I tempi end-to-end riportati in tabella 2 includono latenza di rete e I/O, in linea con quanto osservato per servizi REST/WPS, dove la comunicazione client-server può dominare il costo totale se i payload sono frequenti e piccoli. Questo aspetto è reso esplicito nel protocollo di misura e nella discussione dei trade-off architetturali [Castronova et al., 2013].

Per integrare più tecniche di misura è talvolta necessario assegnare pesi diversi ai dataset. Inoltre, i metodi di esplorazione del parametro (ad es. Neighborhood Algorithm, NA, e campionamento MCMC – *Markov Chain Monte Carlo*, emcee) differiscono per strategia di campionamento e quindi per misfit ottenuto e numero di campioni richiesti. Di conseguenza variano sia i tempi di esecuzione sia la dispersione delle stime dei parametri, come riportato in letteratura [Trasatti, 2022].

4. Possibili applicazioni / integrazioni

Questa sezione illustra scenari d'uso e integrazioni operative per MorphoVolc DEM Interface, concentrandosi sul monitoraggio morfometrico, l'analisi preliminare rapida su DEM a 30 m e le esecuzioni in *batch* tramite API. La struttura si ispira ai report applicativi, adattandoli al dominio vulcanologico.

Integrazione in flussi di lavoro e sistemi di supporto alle decisioni. L'esposizione delle funzionalità come servizi REST locali (con eventuale adattamento a profili OGC/WPS non implementati in questa versione) ne facilita l'integrazione in ambienti e editor di flussi di lavoro (ad esempio, basati su OpenMI). Questo approccio favorisce il riuso e la componibilità delle funzioni all'interno di catene di elaborazione e sistemi di supporto alle decisioni [Castronova et al., 2013].

Scalabilità e supporto multiutente. L'integrazione con framework lato server come Actinia consente di creare ambienti multiutente, gestire code di elaborazione e garantire la tracciabilità asincrona dei processi. Questa metodologia è già stata sperimentata con successo nella piattaforma OpenPlains per la modellazione partecipativa in cloud [White et al., 2023].

Analisi morfometrica preliminare e valutazione della pericolosità. La disponibilità di contorni e volumi standardizzati fornisce valori di riferimento utili per valutare l'instabilità dei versanti e i rischi correlati. Ciò si inserisce in continuità con l'utilizzo di strumenti come MORVOLC e NETVOLC in studi internazionali e conferma l'applicabilità dell'approccio MorphoVolc anche negli scenari di valutazione della pericolosità vulcanica [Zorn et al., 2022].

Accoppiamento con inversione geodetica. MorphoVolc può essere combinato con strumenti open source di inversione geodetica (es. VSM): i contorni/volumi diventano feature ad alto livello da integrare con InSAR (Interferometric Synthetic Aperture Radar) / GNSS (Global Navigation Satellite System), sfruttando portafogli di sorgenti analitiche (Mogi, sferoide, dislocazione) e campionamento Bayesiano o globale per stime rapide e tracciabili [Trasatti, 2022].

4.1 Monitoraggio morfometrico operativo

Questa funzionalità fornisce indicatori semplici e ripetibili — aree, assi principali, rapporti di forma e stime volumetriche — per serie temporali di DEM/DSM (Digital Surface Model) post-evento, standardizzando delimitazioni e criteri di calcolo tra acquisizioni e team diversi. Gli output standardizzati (GeoJSON, CSV, PDF) permettono di confrontare nel tempo rilievi e acquisizioni eterogenee con criteri omogenei e tracciabili. Nel presente contesto, “monitoraggio” indica soprattutto la possibilità di produrre in modo coerente contorni e metriche a scala di edificio, con pacchetti di output confrontabili; su DEM a scala regionale (~30 m) l'uso tipico è lo screening e la comparazione di trend macroscopici, non la misura di micro-variazioni topografiche.

Quando sono disponibili DEM pre- e post-evento correttamente co-registrati, il confronto differenziale tramite differenza tra modelli digitali di elevazione (DEM of Difference, DoD) rappresenta l'approccio più diretto per mappare variazioni locali e derivare mappe di variazione. In tale scenario, MorphoVolc si colloca come strumento complementare, utile per standardizzare contorni/metriche e produrre indicatori sintetici (Δ area, Δ volume analitico) e un bundle tracciabile di risultati.

Componenti chiave:

- L'impronta basale e l'asse principale (indicatore della tendenza all'allungamento/ellitticità della struttura).
- L'area e il volume netto (calcolato come [volume dell'edificio] – [volume della caldera]) come indicatori sintetici per valutare variazioni nel tempo (crescita/erosione) in modo coerente con la delimitazione adottata.

Un pacchetto completo di prodotti (panoramiche/pannelli della fase /process e, se richiesti, derivati singoli), contorni e un rapporto di esecuzione garantisce tracciabilità e verificabilità dell'analisi.

Operatività:

- Aggiornamento periodico su nuove scene (DEM a scala regionale, es. 30 m; campagne UAS/airborne quando disponibili).
- Confronto tra situazione di riferimento e post-evento (previa co-registrazione/armonizzazione dei dati), con esportazione delle mappe di variazione (Δ area, Δ volume) e, quando applicabile, prodotti differenziali DoD.

4.2 Screening rapido su DEM 30 m

Uso tipico per prioritizzare sopralluoghi o analisi ad alta risoluzione. Con 30 m di passo, la *pipeline* eroga in scenario tipico entro pochi secondi su $\sim 600 \times 600$ px (fino a $\sim 1200 \times 1200$ px con picchi < 700 MB, cfr. § 5.2).

Output attesi:

- Set minimo: ombreggiatura, pendenza, curvatura; contorni base/caldera; tabella metriche (CSV).
- Opzionale: report PDF con miniature e sommario parametri.

Criteri di lettura rapida.

- Rapporto tra assi principali (anisotropia).
- Δ volume tra modelli (circolare vs ellittico) come indicatore di eccentricità.

A questa scala, i risultati sono intesi come valori di riferimento comparabili (screening) e non come stime di dettaglio per variazioni sub-pixel o depositi sottili.

4.3 Batch via API / headless

In v0.1.1 l'esecuzione batch avviene tramite gli endpoint REST locali esposti dal container, orchestrabili con script/cron e docker compose.

Per lavorazioni massive si impiega la modalità *headless* e le rotte locali. Il riuso del `processId` riduce I/O superfluo e consente lancio multi-job controllato (es. per DEM regionali o time-series).

Pattern consigliati:

- Orchestrazione mediante un programmatore di processi (ad esempio, cron) e una coda locale di `processId`.
- Logging strutturato (JSON, JavaScript Object Notation) con prefissi [TIMING]/[MEMORY] di tempi di esecuzione e utilizzo di memoria per il controllo qualità e la pianificazione della capacità.
- Esportazione di un file di descrizione (manifest) con relativo checksum (opz.) per garantire la tracciabilità della provenienza dei dati.

Scenari:

- Ricalcolo periodico su nuove release di DEM (ASTER/SRTM) o mosaici aggiornati.
- Elaborazione a lotti per siti multipli (inventari di edifici).

4.4 Integrazione nei flussi di lavoro degli osservatori

La soluzione può essere inserita nella catena informativa di una sala operativa come servizio interno (on-premise) o *container* isolato. I prodotti finali memorizzati sul filesystem (nella

cartella `outputs/{processId}/...` sono concepiti per essere indicizzati da strumenti interni o sincronizzati con archivi istituzionali.

Punti di integrazione:

- *Dashboard* interne: collegamenti diretti ai pacchetti di esecuzione e ai report in formato PDF.
- Sistemi di ticket/briefing: allegato CSV/PDF per briefing rapidi.
- Cataloghi dati: harvesting di GeoJSON/CSV con metadati minimi (data/ora, versione env, parametri).

4.5 Didattica e ricerca

L'interfaccia web-first e la trasparenza dei parametri la rendono adatta a laboratori universitari e a progetti di ricerca su morfometria comparata. Le unità e formule esplicitate favoriscono replicabilità e peer-review.

Esempi:

- Esercitazioni su estrazione contorni e stima volumetrica con *dataset* di esempio.
- Valutazione di soglie alternative/adattive e propagazione d'incertezza (*roadmap*).

4.6 Interoperabilità e *roadmap* di estensioni

Interoperabilità:

- Formati aperti: GeoJSON, CSV, GeoTIFF, PNG/JPG; REST locale documentabile.
- Integrazione con altri tool GIS (QGIS/GRASS) via file-based e CLI (Command-Line Interface).

Roadmap:

- Soglie adattive (knee-point, max-slope, Otsu) + bande d'errore su V.
- Opzioni performance (*rugosità* opzionale, filtri separabili/Numba/CuPy + caching layer intermedi).
- Geometrie complesse (multi-lobo, coste/maschere acqua) e QA automatico.

5. Conclusioni e Sviluppi futuri

5.1 Sintesi dei risultati

MorphoVolc DEM Interface rappresenta un contributo operativo e metodologico concreto per la morfometria vulcanica, concepito come piattaforma web-native open source, con release e istruzioni disponibili nel repository associato. L'architettura integrata (React + Node/Express + Python) consente di eseguire analisi morfometriche e volumetriche direttamente da DEM, con un flusso di lavoro standardizzato e output tracciabili, senza richiedere dipendenze esterne o setup complessi.

I casi studio su Miyakejima e Izu-Ōshima hanno dimostrato la robustezza operativa della pipeline e la consistenza interna dei risultati tra modelli circolari ed ellittici, con tempi compatibili con l'uso interattivo (dell'ordine di pochi secondi su ~600×600 px; cfr. § 5.2 per casi più grandi). L'interfaccia è stata validata anche in modalità batch, mostrando riproducibilità elevata grazie al `processId`, al manifest degli artefatti e ai log `[TIMING]/[MEMORY]` (oltre a eventuali parametri/versioni quando configurati).

La versione oggetto della presente release (v0.1.1) è distribuita in forma dockerizzata e fornisce una base stabile e riproducibile per l'esecuzione del workflow. Le istruzioni operative per l'avvio

e la prova dell'interfaccia sono riportate nel box “Disponibilità e prova dell'applicativo” e nel README della release.

Confronti quantitativi su casi reali con stime indipendenti (ad es. DEM pre/post evento o volumi riportati in letteratura) sono previsti come attività di benchmark nell'evoluzione dell'applicativo, insieme a un progressivo affinamento dell'approccio metodologico.

5.2 Limiti attuali

Le principali limitazioni riguardano l'uso di soglie fisse per base (5 %) e caldera (80 %), soluzioni semplici e comparabili ma non adattive; l'altezza del tronco di cono è inoltre calcolata rispetto all'elevazione massima del DEM (h_{max}) e non alla quota media della base. Incide poi il costo computazionale dell'operatore di rugosità, che tende a dominare il tempo di calcolo delle derivate. Infine, l'approccio è attualmente pensato per edifici con rilievo positivo e geometrie regolari, risultando meno adatto a strutture fissurali o a maar.

5.3 Sviluppi futuri (*roadmap*)

La *roadmap* prevede l'introduzione di soglie adattive basate su criteri *data driven* (knee point, max slope, Otsu) e di un'opzione per l'altezza relativa alla base ($h = z_{max} - z_{base_media}$). È pianificata la quantificazione dell'incertezza su volume e area tramite bootstrap o simulazioni Monte Carlo, insieme a ottimizzazioni prestazionali che includono parallelizzazione e caching dei layer intermedi. Sul fronte dell'operatività, è prevista una distribuzione *full-stack* (frontend + backend + Python) per ambienti osservatorio, il supporto a geometrie più complesse (multi-lobo, coste/maschere d'acqua) con QA automatico dei contorni e, infine, l'integrazione di log strutturati (JSON) per il tracciamento delle operazioni e la pianificazione della capacità di calcolo. Validazione e scalabilità. I test riportati nel presente Rapporto Tecnico sono condotti su DEM a ~30 m (SRTM/ASTER) e su matrici nell'ordine di ~600×600 px, con un minibenchmark che quantifica runtime e picchi di memoria (§3.2). La trattabilità di raster più estesi dipende principalmente dalle risorse disponibili (RAM/tempo) e dal costo delle operazioni raster, in particolare nella fase di calcolo delle derivate (es. rugosità/curvature; §3.2). Una caratterizzazione sistematica su DEM a risoluzioni e dimensioni crescenti è pianificata come attività di validazione (cfr. §5.3).

Validazione su casi reali, multi-risoluzione e benchmark con riferimento indipendente. È pianificata una campagna di validazione articolata su due assi complementari.

Contesto e baseline dei dati. La release v0.1.1 è progettata per operare in modo riproducibile su DEM open a copertura ampia e omogenea (tipicamente ~30 m, es. SRTM/ASTER o equivalenti), che rappresentano lo standard più accessibile e comparabile per analisi multi-sito e screening regionale. DEM a risoluzione più alta possono migliorare la qualità dei contorni e delle metriche, ma la loro disponibilità open è spesso regionale/non uniforme o soggetta a condizioni di accesso differenti; per questo la validazione multi-risoluzione è trattata come milestone dedicata.

Sensibilità alla risoluzione e alla qualità del DEM: confronto sistematico tra DEM a ~30 m (es. SRTM/ASTER) e prodotti ad alta risoluzione (LiDAR e/o fotogrammetria/UAS dove disponibili), per quantificare l'effetto di risoluzione/rumore sulla stabilità di contorni (base/caldera), assi principali, aree e stime volumetriche. In questo contesto, le indicazioni metodologiche e le derivate DEM-based discusse in [Favalli e Fornaciai, 2017] forniscono un riferimento utile per impostare confronti coerenti tra parametri derivati e scale diverse.

Benchmark su casi con stime indipendenti di volume/variazione: selezione di casi studio in cui siano disponibili vincoli esterni o confronti pre/post-evento, per valutare scostamenti e condizioni di applicabilità delle stime (inclusi scenari di evoluzione rapida e analisi su edifici monogenetici).

A titolo di riferimento, verranno considerati: (a) casi LiDAR e approcci pre/post per l'evoluzione a breve termine di coni scoriacei su Etna [Fornaciai et al., 2010]; (b) analisi morfometriche DEM-based e relazioni morfologia-contesto geodinamico su popolazioni di coni [Fornaciai et al., 2012] e casi ad alta risoluzione su coni di fianco vulcanico [Favalli et al., 2009]; (c) studi di delimitazione semi-automatica e validazione di contorni di edifici vulcanici [Di Traglia et al., 2014].

L'obiettivo è definire metriche di accuratezza (es. stabilità dei contorni, variazioni percentuali su area/assi/volume, sensibilità alle soglie e ai derivati) e stime di incertezza, distinguendo chiaramente gli scenari in cui l'approccio è più robusto (screening regionale, edifici regolari) da quelli che richiedono integrazioni o controlli aggiuntivi.

5.4 Prospettive

Nel medio termine, MorphoVolc può essere integrato nei flussi di lavoro dell'INGV e di altre istituzioni vulcanologiche per analisi e confronto morfometrico standardizzato di edifici vulcanici nel tempo, in particolare a scala regionale su DEM disponibili (ad es. ~30 m) e, quando disponibili, su prodotti a maggiore risoluzione (fotogrammetria/UAS, LiDAR, InSAR-derived DEM, satelliti ottici ad alta risoluzione). La struttura modulare e i formati aperti (GeoJSON, CSV, GeoTIFF, PNG/PDF) ne facilitano l'inclusione in dashboard, portali di dati o archivi interoperabili.

L'approccio trasparente e parametrico rende inoltre il sistema adatto alla didattica universitaria e alla *citizen science*, favorendo la diffusione di pratiche di *open science* e la replicabilità delle analisi. Gli sviluppi futuri potranno includere l'integrazione con librerie spettrali, strumenti di *machine learning* per la classificazione morfologica e moduli di analisi multi-temporale per lo studio dell'evoluzione volumetrica degli edifici vulcanici.

In sintesi, MorphoVolc riduce tempi e variabilità operative, accorciando il passaggio dai DEM a risultati morfometrici standardizzati con report e tracciabilità integrati. L'adozione di ambienti preconfigurati (Docker/Conda), *dataset* di esempio e test di unità/integrazione assicura verificabilità e riuso multiplatforma, coerenti con i principi FAIR (Findable, Accessible, Interoperable, Reusable) e con l'*open science*.

Dichiarazione software

Il codice sorgente, la release dockerizzata v 0.1.1 e le istruzioni di utilizzo sono disponibili nel repository GitHub indicato nel box "Disponibilità e prova dell'applicativo". Il software utilizza librerie open source di terze parti distribuite secondo le rispettive licenze. La distribuzione non include componenti soggette a restrizioni di ridistribuzione oltre alle licenze delle dipendenze indicate.

Disponibilità e prova dell'applicativo (v0.1.1, Docker edition)

La versione descritta nel presente Rapporto Tecnico è disponibile come v0.1.1 dockerizzata ed è utilizzabile in locale tramite interfaccia web (browser). La distribuzione via Docker garantisce riproducibilità, portabilità e un ambiente isolato, senza installazione manuale di Python/Node.js o librerie scientifiche.

Per la prova dell'applicativo è richiesto solo Docker Desktop. Il progetto può essere scaricato da GitHub (source.zip o repository) e avviato dalla cartella contenente docker-compose.yml con: `docker compose up -build`.

A esecuzione avviata, l'interfaccia è accessibile all'indirizzo: <http://localhost:5000>.

Il workflow tipico prevede: caricamento di un DEM in formato GeoTIFF (.tif), selezione del modulo (base circolare/ellittica; approx1/approx2) ed esecuzione del calcolo con visualizzazione dei risultati numerici, dei prodotti grafici e generazione del report PDF.

Ogni esecuzione genera una cartella backend/outputs/{processId}/ contenente i principali artefatti (es. volume_results.json, analysis_images.json, immagini PNG intermedie e report PDF) e consultabile anche via browser (/outputs/{processId}/).

Codice sorgente, istruzioni aggiornate e dettagli tecnici sono disponibili nel repository GitHub: <https://github.com/Solly87INGV/DEM-Volcano-Analysis/releases/tag/v0.1.1>

Bibliografia

- Castronova, A.M., Goodall, J.L., & Elag, M.M. (2013). Models as web services using the Open Geospatial Consortium (OGC) Web Processing Service (WPS) standard. *Environmental Modelling & Software*, 41, 72–83. <https://doi.org/10.1016/j.envsoft.2012.11.010>
- Di Traglia, F., Morelli, S., Casagli, N., & Monroy, V.H.G. (2014). Semi-automatic delimitation of volcanic edifice boundaries: Validation and application to the cinder cones of the Tancitaro–Nueva Italia region (Michoacán–Guanajuato Volcanic Field, Mexico). *Geomorphology*, 219, 152–160. <https://doi.org/10.1016/j.geomorph.2014.05.002>
- Euillades, L.D., Grosse, P., & Euillades, P.A. (2013). NETVOLC: An algorithm for automatic delimitation of volcano edifice boundaries using DEMs. *Computers & Geosciences*, 56, 151–160. <https://doi.org/10.1016/j.cageo.2013.03.011>
- Favalli, M., & Fornaciai, A. (2017). Visualization and comparison of DEM-derived parameters: Application to volcanic areas. *Geomorphology*, 290, 69–84. <https://doi.org/10.1016/j.geomorph.2017.02.029>
- Favalli, M., Karátson, D., Mazzarini, F., Pareschi, M.T., & Boschi, E. (2009). Morphometry of scoria cones located on a volcano flank: A case study from Mt. Etna (Italy), based on high-resolution LiDAR data. *Journal of Volcanology and Geothermal Research*, 186(3–4), 320–330. <https://doi.org/10.1016/j.jvolgeores.2009.07.011>
- Fornaciai, A., Behncke, B., Favalli, M., Neri, M., Tarquini, S., & Boschi, E. (2010). Detecting short-term evolution of Etnean scoria cones: A LiDAR-based approach. *Bulletin of Volcanology*, 72(10), 1209–1222. <https://doi.org/10.1007/s00445-010-0394-3>
- Fornaciai, A., Favalli, M., Karátson, D., Tarquini, S., & Boschi, E. (2012). Morphometry of scoria cones and their relation to geodynamic setting: A DEM-based analysis. *Journal of Volcanology and Geothermal Research*, 217, 56–72. <https://doi.org/10.1016/j.jvolgeores.2011.12.012>
- Krishnan, S., Nandigam, V., Crosby, C.J., Phan, M., Cowart, C., Baru, C., & Arrowsmith, R. (2011). OpenTopography: A services oriented architecture for community access to LiDAR topography. In *Proceedings of the 2nd International Conference on Computing for Geospatial Research & Applications (COM.Geo '11)* (pp. 1–8). <https://doi.org/10.1145/1999320.1999327>
- Qiao, X., Li, Z., Ames, D.P., Nelson, E.J., & Swain, N.R. (2019). Simplifying the deployment of OGC web processing services (WPS) for environmental modelling—Introducing Tethys WPS Server. *Environmental Modelling & Software*, 115, 38–50. <https://doi.org/10.1016/j.envsoft.2019.01.021>
- Trasatti, E. (2022). VSM: A Python tool for volcano geodetic source modeling via Voronoi and Bayesian approaches. *Computers & Geosciences*, 165, 105162. <https://doi.org/10.1016/j.cageo.2022.105162>

- White, C.T., Petrasova, A., Petras, V., Tateosian, L.G., Vukomanovic, J., Mitasova, H., & Meentemeyer, R.K. (2023). An open-source platform for geospatial participatory modeling in the cloud. *Environmental Modelling & Software*, 167, 105767. <https://doi.org/10.1016/j.envsoft.2023.105767>
- Zorn, E.U., Orynbaikyzy, A., Plank, S., Babeyko, A., Darmawan, H., Robbany, I.F., & Walter, T.R. (2022). Identification and ranking of subaerial volcanic tsunami hazard sources in Southeast Asia. *Natural Hazards and Earth System Sciences*, 22, 3083–3104. <https://doi.org/10.5194/nhess-22-3083-2022>

APPENDICI

Appendice A

Hook di *polling* lato interfaccia

Questa appendice presenta un breve *hook* React (`useProcessStatus`) che esegue il *polling* dello stato con un leggero back-off, aggiorna in tempo reale l'interfaccia e si arresta al completamento. Supporta esecuzioni parallele (*multi-run*).

```
// useProcessStatus.js – hook di polling con backoff lineare
import { useEffect, useRef, useState } from "react";
import axios from "axios";

export function useProcessStatus(processId) {
  const [status, setStatus] = useState(processId ? "processing" : "idle");
  const [elapsed, setElapsed] = useState(0);
  const timerRef = useRef(null);
  const startTsRef = useRef(null);

  useEffect(() => {
    if (!processId) return;
    let tick = 0;
    startTsRef.current = Date.now();

    const poll = async () => {
      try {
        const { data } = await axios.get(`/processStatus/${processId}`);
        setStatus(data.status);
        setElapsed(Math.floor((Date.now() - startTsRef.current) / 1000));
        if (data.status === "processing") {
          tick += 1;
          const delay = Math.min(2000 + tick * 250, 5000); // backoff 2s→5s
          timerRef.current = setTimeout(poll, delay);
        }
      } catch {
        // in caso di errore rete riprova blandamente
        timerRef.current = setTimeout(poll, 3000);
      }
    };

    poll();
    return () => clearTimeout(timerRef.current);
  }, [processId]);

  return { status, elapsed, isDone: status === "completed" || status === "error" };
}
```

Appendice B

Endpoint POST /calculateVolume

Orchestrazione Node/Express-Python per la selezione dello script volumetrico e ritorno del risultato in JSON.

```
// Endpoint: POST /calculateVolume
app.post('/calculateVolume', upload.single('demFile'), (req, res) => {
  const { volumeType, approximationType } = req.body;
  const file = req.file;
  if (!file || !volumeType || !approximationType)
    return res.status(400).json({ error: "Missing required fields." });

  // Selezione dello script Python in base al modello richiesto
  let scriptPath;
  if (volumeType === 'circular')
    scriptPath = path.join(__dirname, 'scripts',
      approximationType === 'approximation1' ? 'CircularVolcano_Approx1.py'
      : 'CircularVolcano_Approx2.py');
  else if (volumeType === 'elliptical')
    scriptPath = path.join(__dirname, 'scripts',
      approximationType === 'approximation1' ? 'EllipticalVolcano_Approx1.py'
      : 'EllipticalVolcano_Approx2.py');

  // Avvio processo Python
  const child = spawn(pythonPath, [scriptPath, file.path], {
    stdio: ['pipe', 'pipe', 'pipe'],
    env: { ...process.env, PYTHONUNBUFFERED: '1' }
  });

  let resultData = '';
  child.stdout.on('data', d => resultData += d.toString());
  child.stderr.on('data', d => console.error(`[PY ERR] ${d}`));

  child.on('close', code => {
    if (code === 0) res.json({ result: resultData });
    else res.status(500).json({ error: 'Error calculating volume' });
  });
});
```

Appendice C

Endpoint di avvio e gestione del processo

Estratto dell'endpoint Node/Express POST /process: riceve il DEM, genera l'ID «processId», avvia Python *headless* e salva log/artefatti per il *polling* della UI.

```
// POST /process - upload DEM → crea processId → lancia Python headless
import { Router } from "express";
import multer from "multer";
import { v4 as uuidv4 } from "uuid";
import { spawn } from "child_process";
import path from "path";
import fs from "fs";

const router = Router();
const upload = multer({ dest: "uploads/" });

// stato volatile: { [processId]: "processing" | "completed" | "error" }
const JOB_STATE = Object.create(null);

router.post("/process", upload.single("demFile"), (req, res) => {
  const processId = uuidv4();
  JOB_STATE[processId] = "processing";

  const demPath = req.file.path; // file tmp caricato
  const outDir = path.join("outputs", processId);
  fs.mkdirSync(path.join(outDir, "logs"), { recursive: true });

  const py = spawn(
    "python",
    ["complete_dem_analysis.py", "--dem", demPath, "--out", outDir],
    {
      env: {
        ...process.env,
        MPLBACKEND: "Agg",
        QT_QPA_PLATFORM: "offscreen",
        OPENBLAS_NUM_THREADS: "1",
      },
      stdio: ["ignore", "pipe", "pipe"],
    }
  );

  const logOut = fs.createWriteStream(path.join(outDir, "logs", "stdout.log"));
  const logErr = fs.createWriteStream(path.join(outDir, "logs", "stderr.log"));
  py.stdout.pipe(logOut);
  py.stderr.pipe(logErr);

  py.on("exit", (code) => {
    JOB_STATE[processId] = code === 0 ? "completed" : "error";
  });

  // risposta immediata: il client inizia il polling
  res.status(202).json({ processId });
});

// GET /processStatus/:id - ritorna stato correntemente noto
router.get("/processStatus/:id", (req, res) => {
  const st = JOB_STATE[req.params.id] || "error";
  res.json({ status: st });
});

export default router;
```

Appendice D

Delimitazione di base e caldera e scelta dei punti opposti

Esempio in NumPy/scikit-image per: (i) contorno di base a $z_{\text{base}} = z_{\text{min}} + 0,05 (z_{\text{max}} - z_{\text{min}})$; (ii) contorno di caldera a $z_{\text{cald}} = 0,80 z_{\text{max}}$; (iii) coppie di punti opposti (*antipodali* sulla base; max slope + antipodale sulla caldera).

```
import numpy as np
from scipy.ndimage import sobel
from skimage import measure

def find_lowest_base_contour(matrix, base_elevation_ratio=0.05):
    """Contorno base:  $z_{\text{base}} = z_{\text{min}} + r(z_{\text{max}} - z_{\text{min}})$ ."""
    level = matrix.min() + (matrix.max() - matrix.min()) * base_elevation_ratio
    contours = measure.find_contours(matrix, level)
    if not contours:
        raise ValueError("No contours found for the given base elevation ratio.")
    return max(contours, key=len) # contorno principale

def find_opposite_base_points(contour):
    """Due vertici ~antipodali: indici 0 e N//2."""
    c = np.round(contour).astype(int)
    return tuple(c[0]), tuple(c[len(c)//2]) # (r0, c0), (r1, c1)

def calculate_slope(matrix):
    """Pendenza = modulo del gradiente (Sobel)."""
    dx = sobel(matrix, axis=1)
    dy = sobel(matrix, axis=0)
    return np.hypot(dx, dy)

def find_caldera_contour(matrix, level_ratio=0.80):
    """Contorno caldera:  $z_{\text{cald}} = r \cdot z_{\text{max}}$ ; ritorna il principale."""
    level = matrix.max() * level_ratio
    contours = measure.find_contours(matrix, level)
    if not contours:
        raise ValueError("No contours found for the given level ratio.")
    return max(contours, key=len)

def find_opposite_slope_points(slope_matrix, contour):
    """Su caldera: vertice a pendenza max + opposto a ~N/2."""
    c = np.round(contour).astype(int)
    # filtro limiti matrice
    c = c[(c[:,0] >= 0) & (c[:,0] < slope_matrix.shape[0]) &
          (c[:,1] >= 0) & (c[:,1] < slope_matrix.shape[1])]
    if len(c) == 0:
        raise ValueError("No valid points in contour after filtering.")
    slopes = slope_matrix[c[:,0], c[:,1]]
    i_max = int(np.argmax(slopes))
    i_opp = (i_max + len(c)//2) % len(c)
    return tuple(c[i_max]), tuple(c[i_opp])
```

Appendice E

Troubleshooting rapido

Scopo. Elenco non esaustivo di problemi comuni, con causa probabile e azione correttiva.

- Errore: “CRS mancante / EPSG non valido”

Causa: DEM senza sistema di riferimento o codice EPSG errato.

Soluzione: definire/validare il CRS (es. EPSG:4326 o UTM corretto); rigenerare il GeoTIFF con tag geodetici coerenti.

- DEM rifiutato / formato non riconosciuto

Causa: file non GeoTIFF, compressioni insolite, pixel non quadrati.

Soluzione: convertire a GeoTIFF non compresso, pixel square, dtype e nodata definiti (gdal_translate).

- Macchie bianche / bordi vuoti nelle immagini

Causa: valori NoData non dichiarati o maschere incoerenti.

Soluzione: impostare NoData esplicito e allineare la maschera prima dell'upload; verificare in §2.2 le derivate.

- Arresto per “MemoryError” o picchi RAM elevati

Causa: dimensioni raster elevate, roughness costosa, ambienti con poca memoria.

Soluzione: ridurre risoluzione/estensione, disabilitare roughness se non necessaria, assicurare OPENBLAS_NUM_THREADS=1.

- Figure non generate in headless

Causa: backend grafico non impostato.

Soluzione: verificare variabili: MPLBACKEND=Agg, QT_QPA_PLATFORM=offscreen (§2.2.2).

- Conflitti rasterio/GDAL

Causa: versioni binarie non allineate.

Soluzione: usare environment dedicato (Conda/conda-forge) o wheel precompilate coerenti; attenersi alle versioni minime di §2.7.4.

- Permessi / salvataggio su outputs/

Causa: directory non scrivibile o path con caratteri speciali.

Soluzione: garantire permessi in outputs/{processId}/, evitare spazi/accents in path, verificare quota disco.

- Processo non “completa” (frontend resta in attesa)

Causa: mancata callback POST /processComplete/{processId} o crash script.

Soluzione: controllare log con prefissi [TIMING]/[MEMORY], stderr del child process, e lo stato in manifest.json.

- PDF incompleto / immagini mancanti

Causa: asset non trovati da ReportLab o nomi file non standard.

Soluzione: verificare che dem_overview.png, double_*, triplet_* esistano e seguano la convenzione (§2.5.1).

- Tempi eccessivi su serie di job piccoli

Causa: latenza I/O e chiamate REST frequenti.

Soluzione: riuso del processId, batching locale, caching dei layer intermedi (§3.2).

(Per problemi non elencati: verificare le versioni runtime in §2.7.1 e le dipendenze minime in §2.7.4).



QUADERNI di GEOFISICA

ISSN 1590-2595

<https://collane-editoriali.ingv.it/index.php/qdg/it/index/>

QUADERNI di GEOFISICA è una rivista scientifica che pubblica dati, misurazioni e osservazioni geofisiche (anche preliminari) in italiano e inglese. L'obiettivo è una diffusione rapida e gratuita per la comunità scientifica, garantita da un processo di peer-review e da un Editorial Board multidisciplinare. La rivista è inclusa in "Emerging Sources Citation Index" di Clarivate Analytics e "Open Access Journals" di Scopus, e adotta il modello Diamond Open Access, che non prevede costi né per gli autori né per i lettori.

QUADERNI di GEOFISICA is a scientific journal that publishes geophysical data, measurements and observations, including preliminary results, in Italian and English. Its aim is to ensure rapid and free dissemination to the research community, supported by a peer-review process and by a multidisciplinary Editorial Board. The journal is included in the Emerging Sources Citation Index of Clarivate Analytics and it's included in the Open Access journals of Scopus. It follows the Diamond Open Access model, with no charges for either authors or readers.



RAPPORTI TECNICI INGV

ISSN 2039-7941

<https://collane-editoriali.ingv.it/index.php/rt/it/index>

RAPPORTI TECNICI INGV è una rivista tecnico-scientifica che mette in mostra le numerose attività tecnologiche e di ricerca condotte dai ricercatori e tecnici INGV, e non solo. Pubblicata in italiano e inglese, RAPPORTI TECNICI INGV è ideata per valorizzare anche quei lavori che offrono un contributo significativo dal punto di vista tecnico, operativo o documentale fondamentale per lo sviluppo dell'Ente. La rivista adotta il modello Diamond Open Access, che non prevede costi né per gli autori né per i lettori.

RAPPORTI TECNICI INGV is a technical-scientific journal that showcases the wide range of technological and research activities carried out by INGV researchers and technical staff, as well as by external contributors. Published in Italian and English, RAPPORTI TECNICI INGV is designed to highlight works that provide a significant contribution from a technical, operational, or documentary perspective, and that are essential to INGV development. The journal follows the Diamond Open Access model, with no charges for either authors or readers.



MISCELLANEA INGV

ISSN 2039-6651

<https://collane-editoriali.ingv.it/index.php/mi/it/index>

MISCELLANEA INGV rappresenta una tipologia di editoria non tradizionalmente catalogabile nelle ordinarie riviste scientifiche di settore. Pubblicata in lingua italiana e inglese, MISCELLANEA INGV offre uno spazio per la pubblicazione di materiali eterogenei che, pur non rientrando nei formati più strutturati delle altre Collane editoriali INGV, rivestono un interesse scientifico, tecnico o divulgativo. La rivista adotta il modello Diamond Open Access, che non prevede costi né per gli autori né per i lettori.

MISCELLANEA INGV represents a form of publishing that cannot be classified within the standard scientific journals. Published in Italian and English, MISCELLANEA INGV provides a venue for the publication of heterogeneous materials that are of scientific, technical or outreach interest, even though they do not fall within the more structured formats of the other INGV editorial series. The journal follows the Diamond Open Access model, with no charges for either authors or readers.

Progetto grafico
Barbara ANGIONI
Istituto Nazionale di Geofisica e Vulcanologia

Ufficio Editoriale
Francesca DI STEFANO | Coordinatore

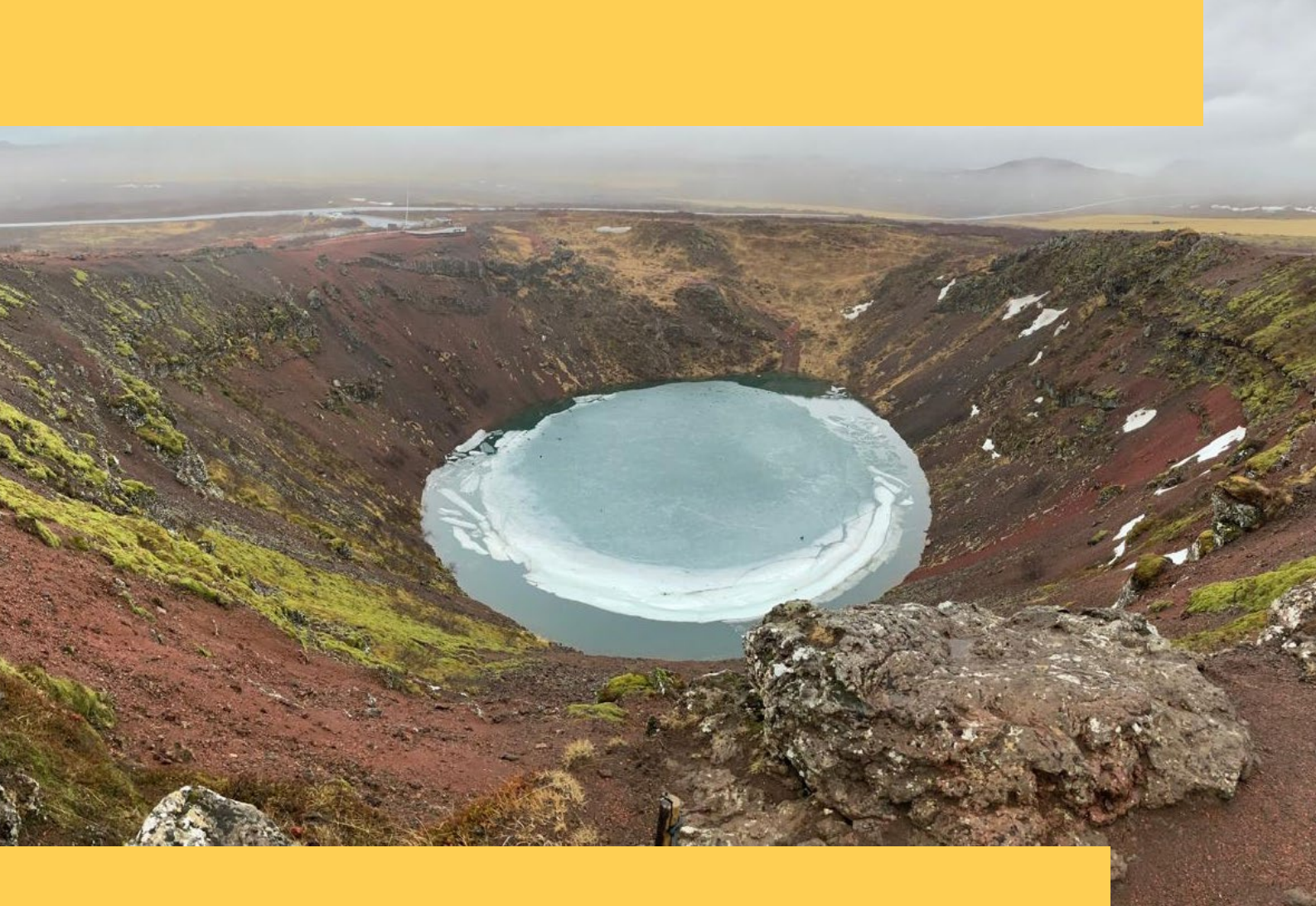
Segreteria di redazione
segreteria.collane-editoriali@ingv.it

Editing | Produzione | Grafica redazionale
Barbara ANGIONI
Massimiliano CASCONI
Francesca DI STEFANO
Patrizia PANTANI

© 2026 the Author(s). Open Access.
This article is licensed under a Creative Commons Attribution 4.0 International License.



Creative Commons Attribution 4.0 International



ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA