PyRadiomics-cuda: a GPU-accelerated 3D features extraction from medical images within PyRadiomics

Jakub Lisowski, Piotr Tyrakowski, Szymon Zyguła, Krzysztof Kaczmarski

Warsaw University of Technology, Poland, 3 October 2025

Abstract

PyRadiomics-cuda is a GPU-accelerated extension of the PyRadiomics library, designed to address the computational chalges of extracting three-dimensional shape features from medl images. By offloading key geometric computations to GPU rdware it dramatically reduces processing times for large lumetric datasets. The system maintains full compatibilwith the original PyRadiomics API, enabling seamless inteation into existing AI workflows without code modifications. is transparent acceleration facilitates efficient, scalable ramics analysis, supporting rapid feature extraction essential high-throughput AI pipeline. Tests performed on a typil computational cluster, budget and home devices prove uselness in all scenarios. PyRadiomics-cuda is implemented in thon and C/CUDA and is freely available under the BSD ense at `https://github.com/mis-wut/pyradiomics-CUDA` ditionally PyRadiomics-cuda test suite is available at `https: github.com/mis-wut/pyradiomics-cuda-data-gen`. It ovides detailed handbook and sample scripts suited for fferent kinds of workflows plus detailed installation inuctions. The dataset used for testing is available at ggle `https://www.kaggle.com/datasets/sabahesaraki/ dney-tumor-segmentation-challengekits-19`

Introduction

AI era, radiomics remains an evolving discipline that seeks extract quantitative features from standard medical images, nsforming visual radiological data into high-dimensional, neable information. This approach supports the development predictive and prognostic models for cancer characterization, treatment response evaluation, and survival analysis. Among the most influential tools in the radiomics domain is PyRadiomics, a widely used, open-source Python library [8]. PyRadiomics provides a framework for the standardized extraction of engineered features from medical imaging data such as CT, MRI, and PET scans. These features include first-order statistics, shape descriptors, and a broad range of texture metrics derived from matrices such as GLCM, GLRLM, and GLSZM[1]. The library's modular architecture, support for both 2D and 3D images, and seamless integration with platforms like 3D Slicer [1] and SimpleITK [5] have contributed to its rapid adoption in academic research and clinical trials. Since its original publication, PyRadiomics has become a cornerstone in the radiomics ecosystem, cited in over 1700 scientific publications across a wide range of applications, from lung and head-and-neck cancer studies to brain tumor segmentation and radiogenomic analysis.

Despite its strengths, one of the computational bottlenecks in PyRadiomics is the extraction of shape-based features, which rely on resource-intensive 3D geometric operations such as mesh generation and pairwise distance calculations in planes and overall shape diameters These operations are especially demanding when applied to large image volumes or high-resolution scans, often resulting in long processing times and reduced scalability which radically slows down the radiomics workflow. For instance, the computation of shape features like mesh volume, surface area, and maximum diameters can involve algorithms with theoretical complexities of $O(n^3)$ or higher, where n is the number of voxels in the Region Of Interest (ROI). This computational overhead can be a significant limitation in high-throughput settings, such as clinical trials or large-scale imaging studies, where rapid feature extraction is essential for timely analysis and decision-making.

To overcome these limitations, we present PyRadiomics-cuda, a GPU-accelerated extension of PyRadiomics that significantly improves the performance of shape feature computation by offloading key components, such as the Marching Cubes algorithm and diameter estimation, to a CUDA-enabled device. This enhancement retains full compatibility with the original PyRadiomics API, allowing seamless integration into existing pipelines while delivering substantial speedups and enabling more efficient high-throughput radiomics studies in time-sensitive settings.

1.1 Related works

Several research groups have explored CUDA-enabled implementations to accelerate radiomics workflows, targeting the bottlenecks inherent in shape and texture feature extraction. One notable contribution is a CUDA-based radiomics pipeline for unsupervised medical image analysis, demonstrating performance gains over CPU-based counterparts [7]. Their approach emphasized parallel computation of texture features using CUDA kernels for high-throughput voxel-wise processing. Additionally, there have been experimental forks of PyRadiomics incorporating GPU acceleration through PyCUDA or CuPy, particularly in research settings where feature extraction needed to scale across thousands of 3D scans. However, these implementations focused only on first-order feature classes (e.g. Grey Level Matrix) and lacked full compatibility with the original PyRadiomics architecture requiring significant restructuring of input pipelines, limiting their general usability.

Another significant tool in this field is cuRadiomics [3], an open-source library designed to replicate and extend

[1] Gray Level Co-occurrence Matrix, Gray Level Run Length Matrix, Gray Level Size Zone Matrix

PyRadiomics functionality on CUDA-enabled hardware. Unlike earlier GPU adaptations, cuRadiomics offers a comprehensive reimplementation of the radiomic feature extraction pipeline, leveraging NVIDIA's CUDA architecture to accelerate a wide spectrum of feature classes, including first-order statistics, shape descriptors, and texture features such as GLCM and GLRLM. It incorporates memory-optimized GPU kernels, stream-based asynchronous execution, and a modular C++/CUDA architecture to support rapid feature extraction. In benchmarking studies, cuRadiomics has demonstrated speedups of 10× to 100× over traditional CPU-based methods. However, this performance gain comes with the trade-off of API divergence, as cuRadiomics does not maintain backward compatibility with PyRadiomics, often requiring tailored input formats and custom integration pipelines. It is also exclusively dedicated to the extraction of first-order statistical features (entropy, energy, etc.) and gray-level matrix features (contrast, entropy, and correlation of the matrix) and does not compute any geometric properties of the input images. Furthermore, cuRadiomics lacks meaningful integration with Python; it is merely an experimental implementation that could potentially be adapted for further use since running and configuring it is non-trivial. It also must be noted that its code repository has not been updated since 2021.

Unlike these prior efforts, PyRadiomics-cuda introduces a tightly integrated CUDA backend that maintains backward compatibility with the standard PyRadiomics API, automatically detecting and leveraging GPU resources for shape-based features without disrupting existing codebases. This design facilitates seamless adaptation in computational processes, which can be applied to large volume data sets for example in artificial intelligence learning processes. Comparing to cuRadiomics, our philosophy differs: cuRadiomics aims for full-GPU reimplementation and maximal speed only in first-order features, whereas PyRadiomics-cuda emphasizes transparent acceleration and modular integration within the original PyRadiomics framework for the 3D shape features.

2 Architecture of the system

PyRadiomics-cuda is a seamlessly integrated high-performance extension of the PyRadiomics library, built to accelerate the extraction of 3D shape features using GPU computation via CUDA. It addresses the critical computational bottleneck encountered when processing large volumetric medical images, particularly in shape-based descriptors which are expensive to compute sequentially. Currently, PyRadiomics-cuda focuses exclusively on accelerating the `Shape` feature class within PyRadiomics, which includes metrics such as: *mesh volume, surface area, maximum 3D diameter and planar diameters (XY, YZ, XZ planes).*

The process of finding the above volumetric features is based on two steps. In the first one, a parallel threads in a novel highly optimized kernel create a mesh of triangles reproducing the 3d shape of the ROI in the image. Resulting vertices are passed to PyRadiomics transparently to compute all the required features using its built-in functions. However, since diameter calculation for 3D shapes is inherently complex (see Table 2), PyRadiomics-cuda also offers a parallel method for this utilizing several levels of complex parallel data gathering operations to assure the best possible performance from GPU device.

The resulting GPU kernels were optimized for performance and memory usage, leveraging shared memory, coalesced memory accesses to minimize latency and various parallel programming techniques. Overall several combinations of implementation improvements had been tested of which the most effective were added to the final version of the tool like proper organization of global memory using Structure of Arrays (SoA) and reduction of global and block-level atomic operations [9].

2.1 PyRadiomics integration

The integration into the PyRadiomics ecosystem is designed to be seamless and transparent to the end-user. This is accomplished through a build-time injection mechanism that preserves full backward compatibility with the original API.

The process is managed by the convenient `setuptools` configuration of the Python package, which has been extended to handle the CUDA compilation toolchain. During installation, the build system automatically detects the presence of the NVIDIA CUDA Compiler (`nvcc`). If found, it compiles the C/CUDA source files of our extension. Before the compilation, within the C extension code, a single line calling the original CPU-based shape computation function is replaced with a call to the custom dispatcher function. Later, at runtime, this dispatcher determines the execution path: it first queries for a CUDA-capable device, if a suitable GPU is available and the driver initializes successfully, the computation is offloaded to the optimized CUDA kernels. If no GPU is found or an error occurs, the dispatcher gracefully falls back to the original CPU implementation, ensuring that the library remains functional on any system. This approach allows users to benefit from GPU acceleration without any changes to their existing code. The usage is therefore the same as in the other PyRadiomics applications and may be as simple as:

```
from radiomics import featureextractor
ext = featureextractor.RadiomicsFeatureExtractor()
res = ext.execute('scan.nii.gz', 'mask.nii.gz') #input
print(res['MeshVolume'], res['SurfaceArea'])
```

3 Efficiency experiments and runtime results

3.1 Test framework

To validate the numerical correctness and rigorously benchmark the performance of various optimization strategies, a standalone test framework was developed in C and CUDA. It allows for the direct comparison of multiple algorithm implementations, including the original CPU baseline and numerous versions of the GPU kernels, each employing different optimization techniques. A key feature of the test harness is its capacity for granular time measurement, profiling not only the total execution time but also the time consumed by distinct computational stages, such as the Marching Cubes mesh generation and the subsequent diameter calculations, independently.

In order to ensure data integrity between the Python environment and the C test framework, a Python script intercepts the data within the PyRadiomics feature extractor just before it is passed to the C computation function. This script serializes the untouched NumPy arrays (image and mask) to disk as `.npy`

files. The C test framework then utilizes NumPy C API headers to parse these files, guaranteeing that the exact binary data is loaded for testing. The parsed data is passed directly to the C function calls, and execution times are precisely measured using high-resolution system timers, facilitating an accurate and reliable comparison of all implemented algorithms.

All the tests included the following pipeline: reading the input images, applying the segmentation masks computing the radiometric features including volume, surface area and diameters.

3.2 Hardware used and testing data set

There were three different machines used: (1) a professional modern GPU cluster built for AI learning, (2) an average desktop computer and (3) an old server machine equipped with a budget GPU designed to enable AI in refurbished hardware. These configurations are displayed in the Table 1. As an input data set we used Kidney Tumor Segmentation Challenge 19 (KITS19) [2]. The complete set contains more than 200 cases. However, we selected 20 samples covering exemplification of the possible input images sizes from 50kB to 9MB and producing from 2700 to 236588 3-dimensional feature vertices.

3.3 Discussion of the results

In Fig. 1 we can see processing times of the KITS19 data set using PyRadiomics and PyRadiomics-cuda on various hardware configurations with a workflow including creation of a 3D shape from given images and ROI and then calculating volumetric features, diameters and area. The time is presented on log-log scale due to the large differences between the processing times for various datasets from less than 1 millisecond to about 2 minutes: 59 milliseconds on PyRadiomics-cuda on H100 GPU compared to 121 seconds for PyRadiomics on Intel Xeon on 236588 vertices and the same input data.

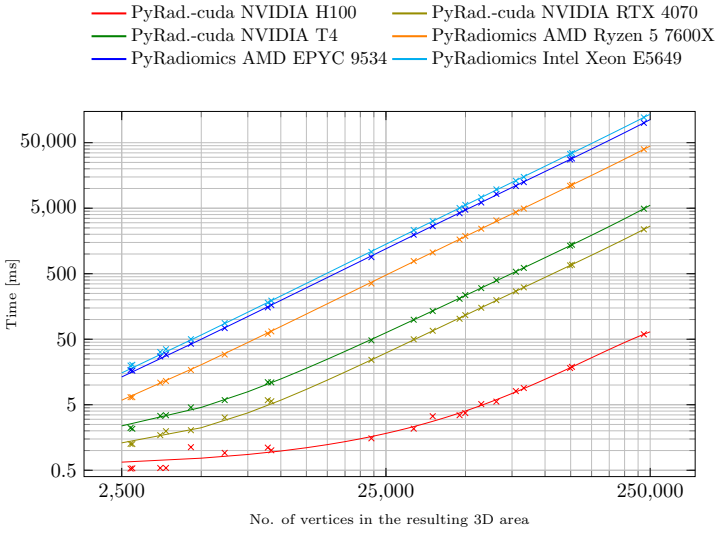

Figure 1: Time efficiency of the KITS dataset feature extraction using PyRadiomics and PyRadiomics-cuda on the various machines. (See Supplementary Information for exact data values of this plot.)

A detailed analysis of the workflow in PyRadiomics indicated that most of the time after reading input file is spent in diameter calculation, from 95.7% for small images to 99.9% for large ones (see Table 2). This is expected due to diameter calculation's theoretical algorithmic complexity of $O(n^4)$. The Marching Cubes algorithm, with a complexity of $O(n^3)$, was significantly faster but still took non-negligible time for large ROIs. Reducing these times was the primary aim of the optimization. Since PyRadiomics is not able to utilize multiple CPU cores, switching to better CPUs lead to only limited speedup, which could be attributed to clock speed and architecture improvements. This can be easily observed in Fig. 2, where switching between CPU configurations 1, 2 and 3 did not reduce the processing time more than 3 times. Using parallel processing on GPUs led to much higher speedup of Marching Cubes and diameter calculation algorithms, which can be attributed to use of massively parallel threads. Even on a budget GPU, PyRadiomics-cuda achieved a speed-up from 8 to 24 times in 3D features extraction. For modern GPUs and more expensive hardware, the processing time was reduced by more than 50 and even 2000 times when compared to our Intel Xeon processor.

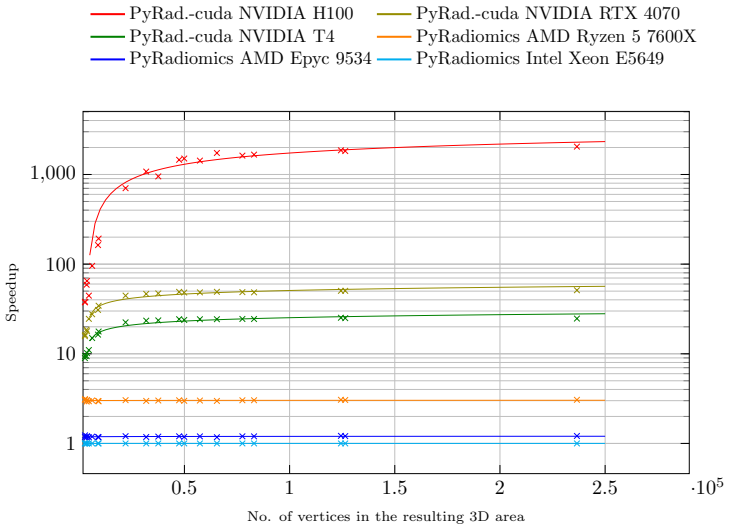

Figure 2: Available speedup of 3D features processing when using PyRadiomics-cuda on various GPU devices compared to the original implementation working on CPU processors. Intel Xeon with PyRadiomics taken as a base reference time. (See Supplementary Information for exact data values of this plot.)

In Table 2 we can also observe that PyRadiomics spends a long time loading data files. This cost comes not only from disk operations but also from various other operations like cleaning, normalization and conversion to proper memory layout. Improving this step still remains an open challenge which could be done on GPU as well.

For complete workflows including data loading and preprocessing utilization of PyRadiomics-cuda resulted in 8 times speed-up for bigger data files with more than 200k vertices in 3D space. Since this is not uncommon case for bigger ROIs, we predict huge savings in processing costs for machine learning tasks on clusters.

4 Conclusions

The initial motivation for this work was long time of feature extraction needed for training AI model in the xLUNGS project [6] dealing with approximately 40000 lungs CT (Computer To-

Table 1: Configurations of the machines used for testing

No.	Machine Type	GPU Device	GPU Cores / VRAM	CPU Model	CPU Cores / Clock / RAM
1	Modern Cluster	NVIDIA H100	14 592 / 80 GB	AMD EPYC 9534	64 / 2.45 GHz / 1 TB
2	Desktop	NVIDIA RTX 4070	5 888 / 12 GB	AMD Ryzen 5 7600x	6 / 5.3 GHz / 32 GB
3	Budget Cluster	NVIDIA T4	2 560 / 16 GB	Intel Xeon E5649	6 / 2.93 GHz / 18 GB

Table 2: The breakdown of the processing time in milliseconds into individual steps for PyRadiomics on CPU processor and PyRadiomics-cuda on GPU processor, both from conf. 2 in Table 1. M.Cubes – Marching Cubes algorithm time, Diameters – 3D diameters calculation time, D. transfer – time needed to copy data from CPU to GPU memory. Representative sample of KITS'19 data set [2]. (Complete table data available in Supplementary Information.)

case no.	**image size**	**vertices in 3D space**	**File [ms] reading**	**PyRadiomics time [ms]**			**PyRadiomics-cuda time [ms]**				**Speedup [x1]**	
				M.Cubes	Diameters	Total	D. transfer	M.Cubes	Diameters	Total	comp.	overall
00000-1	231x104x264	124406	2346	20.702	9516.538	9537.24	8.045	7.179	514.797	530.021	17.994	4.131
00000-2	28x30x59	6132	2350	0.362	25.256	25.618	0.326	0.184	2.418	2.928	8.75	1.01
00001-1	322x126x219	236588	2494	29.451	34210.327	34239.778	9.73	11.006	1855.825	1876.561	18.246	8.404
00001-2	51x62x135	8928	2521	2.274	51.383	53.657	0.683	0.554	3.432	4.668	11.494	1.019
00002-1	230x109x163	83098	1032	13.385	4256.214	4269.599	5.065	4.811	231.794	241.671	17.667	4.162
00002-2	50x45x44	9206	1024	0.577	56.903	57.48	0.453	0.29	3.94	4.683	12.274	1.051
00003-1	237x122x135	77560	1105	12.69	3730.998	3743.688	4.824	4.622	204.071	213.517	17.533	3.677

mography) scans [4], perhaps the world's largest chest X-ray database. Thanks to PyRadiomics-cuda the 3D features extraction time was significantly reduced which led to crucial savings in expensive infrastructure utilization costs. PyRadiomics-cuda proved to deliver the same quality output as original PyRadiomics, but in a fraction of the time not requiring any changes to the existing workflows.

5 Competing interests

No competing interest is declared.

6 Author contributions statement

J.L. and P.T. implemented the tool, created the public repository, testing framework and performed the experiments. J.L., S.Z. and K.K. worked on the text, data analysis and presentation.

7 Acknowledgments

This research was carried out with the support of the High Performance Computing Center at Faculty of Mathematics and Information Science Warsaw University of Technology.

References

[1] Andriy Fedorov, Reinhard Beichel, Jayashree Kalpathy-Cramer, et al. 3D Slicer. `https://www.slicer.org/`, 2025. Accessed: August 22, 2025.

[2] Nicholas Heller, Niranjan Sathianathen, Arveen Kalapara, et al. The KiTS19 Challenge Data: 300 Kidney Tumor Cases with Clinical Context, CT Semantic Segmentations, and Surgical Outcomes, 2020.

[3] Yining Jiao, Oihane Mayo Ijurra, Lichi Zhang, Dinggang Shen, and Qian Wang. cuRadiomics: A GPU-Based Radiomics Feature Extraction Toolkit. In Hassan Mohy-ud Din and Saima Rathore, editors, *Radiomics and Radiogenomics in Neuro-oncology*, pages 44–52, Cham, 2020. Springer International Publishing.

[4] Kamila Kołodko. Polish AI Model Based on the World's Largest Chest X-ray Database. *Research in Poland*, March 2025.

[5] Bradley Lowekamp, David T. Chen, Luis Ibáñez, and Daniel Blezek. SimpleITK. `https://simpleitk.org/`, 2025. Accessed: August 22, 2025.

[6] Marcin Luckner, Paweł Gelar, Agata Kaczmarek, Adam Kaniasty, Bartosz Kochański, et al. Radiomic Medical Data Transformation for Radiologists Support. In *Proceedings of 33rd Int. Conf. on Information Syst. Develop. (ISD2025)*, pages 1–5. AIS, AIS, September 2025.

[7] Leonardo Rundo, Andrea Tangherloni, Paolo Cazzaniga, et al. A CUDA-powered method for the feature extraction and unsupervised analysis of medical images. *The Journal of Supercomputing*, 77(8):8514–8531, August 2021.

[8] Joost JM Van Griethuysen, Andriy Fedorov, Chintan Parmar, et al. Computational radiomics system to decode the radiographic phenotype. *Cancer research*, 77(21):e104–e107, 2017.

[9] Nicholas Wilt. *The CUDA Handbook: A Comprehensive Guide to GPU Programming*. Addison-Wesley Professional, 2013.