buildings

MDPI

Article

Analysis of Concrete Air Voids: Comparing OpenAI-Generated Python Code with MATLAB Scripts and Enhancing 2D Image Processing Using 3D CT Scan Data

Iman Asadi [1,2,*], Andrei Shpak [3] and Stefan Jacobsen [4]

[1] Td-Lab Sustainable Mineral Resources, Department for Knowledge and Communication Management, Faculty of Business and Globalization, University for Continuing Education Krems, 3500 Krems an der Donau, Austria
[2] Built Environment and Engineering Program (BEE), College of Sport, Health, and Engineering (CoSHE), Victoria University, Melbourne, VIC 8001, Australia
[3] Department of Architecture, Building Materials and Structures, SINTEF Community, 7034 Trondheim, Norway; andrei.shpak@sintef.no
[4] Department of Structural Engineering, Norwegian University of Science and Technology, 7491 Trondheim, Norway; stefan.jacobsen@ntnu.no
* Correspondence: iman.asadi@donau-uni.ac.at or iman.asadi@vu.edu.au

Abstract: The air void system in concrete significantly affects its mechanical, thermal, and frost durability properties. This study explored the use of ChatGPT, an AI tool, to generate Python code for analyzing air void parameters in hardened concrete, such as total air void content (A), specific surface (α), and air void spacing factor (L). Initially, Python scripts were created by requesting ChatGPT-3.5 to convert MATLAB scripts developed by Fonseca and Scherer in 2015. The results from Python closely matched those from MATLAB when applied to polished sections of seven different concrete mixes, demonstrating ChatGPT's effectiveness in code conversion. However, generating accurate code without referencing the original MATLAB scripts required detailed prompts, highlighting the need for a strong understanding of the test method. Finally, a Python script was applied to modify void reconstruction in 2D images into 3D by stereology, and comparing this with (3D) CT scanner results, showing comparable results.

Keywords: fly ash; air-entrained concrete; python; cumulative air voids; MATLAB

Citation: Asadi, I.; Shpak, A.; Jacobsen, S. Analysis of Concrete Air Voids: Comparing OpenAI-Generated Python Code with MATLAB Scripts and Enhancing 2D Image Processing Using 3D CT Scan Data. *Buildings* **2024**, *14*, 3712. https://doi.org/10.3390/buildings14123712

Academic Editor: Grzegorz Ludwik Golewski

Received: 15 October 2024
Revised: 10 November 2024
Accepted: 14 November 2024
Published: 21 November 2024

1. Introduction

Concrete contains both intentional and unintentional air voids, each playing a distinct role in its overall performance. Intentional air voids are introduced during the mixing process, often through the addition of air-entraining agents, to improve the concrete's resistance to freeze–thaw cycles [1–3]. These air voids form a network of tiny, uniformly distributed pores, which act as pressure relief zones during freezing. By providing space for ice expansion, these voids reduce internal stress, thereby enhancing durability under fluctuating temperatures [4,5]. In contrast, unintentional air voids result from improper mixing, compaction, or placement and may vary in size and distribution. These irregular voids can weaken the concrete matrix and reduce its overall strength and durability. The distribution, size, and quantity of both intentional and unintentional air voids are crucial in determining the material's performance, especially in harsh environments where freeze–thaw durability is critical. Accurately quantifying these parameters is therefore essential in ensuring concrete's long-term structural integrity and durability [6].

The air void system in concrete is directly assessed by the air void content (%), the specific surface area of air voids (mm^{-1}), and Powers' spacing factor (mm) [7,8]. To achieve high frost durability in concrete, a low average air void spacing factor is desirable, typically less than 0.20 mm. This is because a smaller spacing factor means that the air voids are more evenly distributed and closer together, allowing freezing water to expand into the voids more easily, which helps prevent damage from frost [9].

Image analysis, either in 2D on polished sections or 3D using X-ray tomography, is the primary tool for air void analysis in hardened concrete according to ASTM C457 and its corresponding European version EN 480 [10,11]. Reviews of spacing models and measurements in 2D and 3D based on polished sections and tomography have been given by many researchers, see for example [12–15]. Snyder et al. [13] highlighted that understanding the durability of concrete against freeze–thaw cycles requires a detailed analysis of its air void system. They noted that studying the small air bubbles in concrete typically involves a multi-step process: sample preparation, identification of air voids, detailed analysis of their size and distribution, and assessment of the spacing between them. Each of these steps is supported by research across different engineering disciplines, but civil engineering researchers often need to consult many sources to gather a complete picture. Snyder et al.'s work aims to bring all these aspects together, helping researchers to see the connections, analyze the air void structure effectively, and understand the basis of standard tests like ASTM C 457. Murotani et al. [14] analyzed the air voids in concrete as a 2D spatial point process. They found that the characteristic distance between voids, determined by the nearest neighbor distance function, closely aligned with the traditional spacing factor. By comparing this with a cubic lattice model of air voids (The cubic lattice model of air voids is a conceptual and mathematical model used to represent the spatial distribution of air voids in concrete or other porous materials, and in this model, the air voids are assumed to be regularly spaced and arranged in a three-dimensional cubic lattice structure), they showed that the characteristic distance could serve as an alternative quality parameter. They also introduced a simulation technique to estimate this distance, accounting for the random distribution of voids in cement paste. The results supported the use of point process statistics for evaluating the air void distribution in concrete.

Program coding (computing) plays a vital role in the image analysis of concrete voids, as it does in most engineering fields. It automates the detection and examination of voids, making the process more efficient compared to manual methods. Automated algorithms minimize human error and ensure consistent analysis across different samples and conditions. Through coding, programs can detect and quantify air voids within images, providing detailed metrics such as size, shape, and distribution. The coding also enables image segmentation to isolate voids from the surrounding concrete matrix and calculates statistical properties like void content, specific surface area, and spacing factors. Overall, program coding enhances the accuracy, efficiency, and depth of analysis, facilitating advanced visualization and comprehensive quality control.

The trend of using open-source languages like Python is increasing due to their accessibility, versatility, rich ecosystem, ease of learning, strong community support, industry adoption, educational benefits, and the ability to provide custom solutions. These factors collectively make open-source languages an attractive choice for a wide range of applications and users. Besides increasing the adoption of open-source programming, OpenAI is also a new topic in all research fields. ChatGPT (launched on 30 November 2022) is a newly trained generic AI tool capable of prompting responses and follow-up questions [16]. In February 2023, an article in *Nature* revealed that roughly 80% of the surveyed researchers acknowledged utilizing ChatGPT or similar AI technologies at least once [17].

In this study, Python was specifically chosen because it offers comprehensive image-processing libraries (e.g., OpenCV) and scientific tools (e.g., NumPy, SciPy) that are highly effective for air void analysis in concrete. Python's open-source nature also makes it accessible to a wider audience, allowing researchers without access to MATLAB to replicate and build on our findings. Additionally, OpenAI's ChatGPT, as a tool for Python code generation, enables efficient code development and iteration, making it an asset for rapidly advancing research.

The primary motivations for using ChatGPT varied among respondents. The most common reasons cited included utilizing ChatGPT or similar AI technologies for leisure and entertainment purposes, brainstorming in research endeavors, assistance in coding tasks, aid in creating presentations, preparing literature reviews, drafting research manuscripts, generating visual content, facilitating grant applications, utilizing within scientific research engines, and supporting coursework writing, in that order. Also, many researchers have explored ChatGPT's capabilities in different fields such as public health [18], education [19], global warming [20], psychiatry [21], and the significant impact of its use in technological research [22]. Despite its advantages in several sections, such as its speed for code generation and giving hints for code development, the accuracy of the algorithm is a controversial issue.

A study [23] examined ChatGPT's effectiveness in solving coding problems accurately and efficiently. By testing it with a range of problems from LeetCode, researchers found that ChatGPT achieved correct solutions in about 72% of cases. While it excelled at handling structured tasks, it faced challenges in revising code based on feedback, highlighting a need for better debugging capabilities. Another study [24] explored how AI can improve code generation by using a "self-collaboration" framework, where an AI like ChatGPT takes on different roles—analyst, coder, and tester—to simulate teamwork. They reported that this team-based approach helps AI handle difficult coding projects better than if it worked alone, boosting success rates by around 30–47%. Liu et al. [25] evaluated ChatGPT's ability to generate and fix code across various languages, focusing on correctness, complexity, and security. ChatGPT performed better on older problems but struggled with self-correction, and fixing errors can increase code complexity. Though security issues arose, ChatGPT addressed most vulnerabilities with guidance, pointing to both its strengths and areas for improvement in coding tasks. However, an investigation by Feng et al. [26] reported that ChatGPT is mainly used for languages like Python and JavaScript, but surprisingly, most people feel fearful about using it, rather than happy or excited.

Overall, this work evaluates the potential of OpenAI and Python-based coding approaches to accelerate and improve air void image analysis in hardened concrete. In our study, we utilized ChatGPT's capabilities for code conversion and generation, building upon the experimental results and MATLAB code developed by Fonseca and Scherer. This allowed us to explore how ChatGPT can assist in converting and generating code within the context of our research, providing valuable insights into the evolving field of AI-driven code generation and enhancing the analysis process in concrete research. Finally we compared 3D CT scans of air voids with 3D void size distributions calculated from 2D polished sections using stereology.

2. Materials and Methods

As previously mentioned, the main objective of this study is to develop and assess Python-based code for analyzing air void distributions in concrete, with assistance from OpenAI's ChatGPT for code generation. To fulfill this objective, our methodology consists of the following steps (Figure 1 presents the schematic flowchart of the study):

A. ***Development of Python Scripts:*** We initially converted three existing MATLAB scripts (Fonseca and Scherer [27] (contains three separate processes: (1) reading the image and converting it; (2) basic image analysis based on input from the image conversion and volumetric paste/aggregate ratio of the concrete mix (the scanning can only detect air voids but cannot measure aggregate volume fraction, so the latter is an important input in the analysis); and (3) the code plots 2D void distributions and reconstructs the 3D spheres))—responsible for scanning, calculation, and statistical analysis of air void distributions—into Python versions. This conversion laid the groundwork for a full comparison between MATLAB and Python in performing the analysis.

B. ***Compact Code Generation* via *ChatGPT*:** Using ChatGPT, we aimed to create a streamlined and optimized Python code by merging and compacting the three Python scripts into a single, efficient script. ChatGPT was prompted iteratively to refine the code, aiming for faster processing and easier implementation.

C. ***Evaluation of ChatGPT's Code Generation Capability:*** We evaluated ChatGPT's ability to generate accurate and functional Python code based solely on prompt guidance, without referencing the original MATLAB scripts. This assessment highlights ChatGPT's potential for automating code conversion in scientific applications.

D. ***Concrete Mix Data Source:*** The concrete mix designs and hardened air void data used in this study were sourced from previous research [28,29]. These data served as the foundation for testing and validating the Python code developed.

E. ***Performance Assessment of Python* vs. *MATLAB*:** To assess the effectiveness of the Python code, we compared the air void analysis results produced by Python with those obtained using the original MATLAB scripts by Fonseca and Scherer [27]. This comparison allowed us to determine the accuracy and reliability of the Python-generated results.

F. ***3D Reconstruction Enhancement*:** Lastly, we developed a Python script to compare the stereological results with the CT scanner data. This script introduced a correction factor for improved 3D reconstruction, enhancing the accuracy of air void representation.

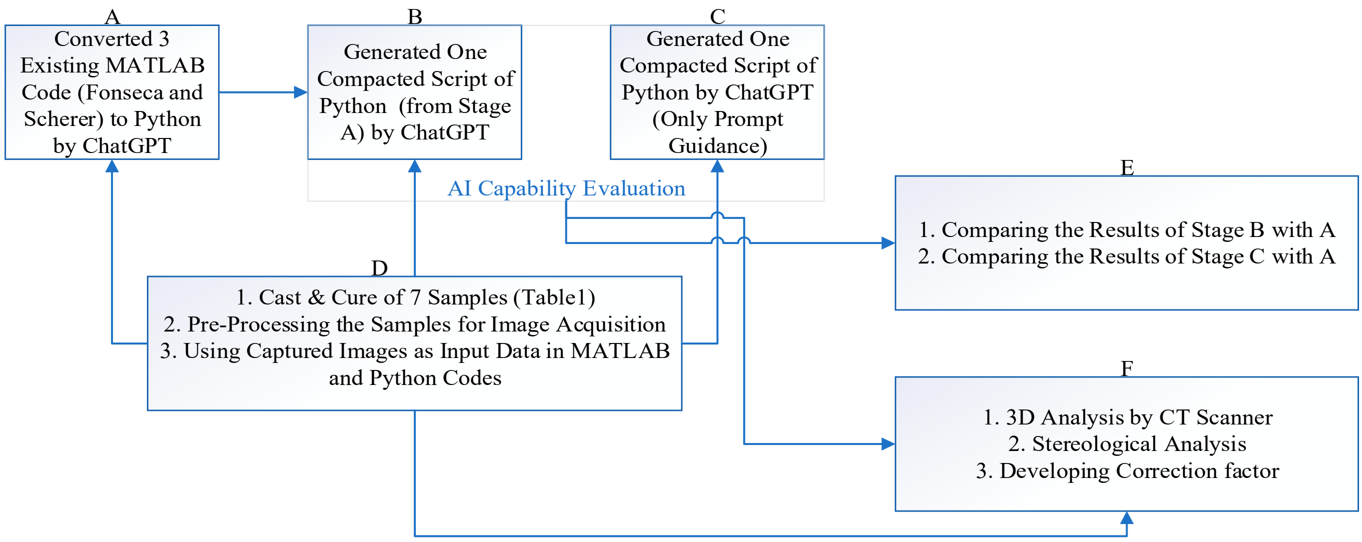


Figure 1. The flowchart of this study.

2.1. Python Script for Air Void Analysis Based on Fonseca and Scherer [27], Own Coding, and ChatGPT

As mentioned, in the first step, the MATLAB scripts were copied to ChatGPT for the conversion of each step into the Python script. After that, we asked ChatGPT to combine these three scripts into a single Python script. During the conversion process, some errors occurred, so we copied the errors into the chat section and requested the corrections. After generating a correct version of code, we went a step further and asked ChatGPT to create a graph showing the cumulative measured voids versus the size of voids. We then requested a stereological analysis to produce a 3D interpretation of the spherical diameter of voids versus the number of detected voids based on the 2D image. Finally, we prepared detailed prompts for ChatGPT to generate code for image analysis to detect the air void system (Appendix A).

2.2. Air Void Data

Previous studies have shown that fly ash concrete requires more air-entraining agent (AEA) due to unburned carbon, which reduces the effectiveness of the AEA [30,31]. Achieving frost-resistant fly ash (FA) concrete with a stable air void system remains challenging because of the variable carbon content in fly ash and other factors like cenospheres and plerospheres that adsorb the AEA [32]. Common practices of increasing AEA dosage do not account for these variations, leading to inconsistent air entrainment and necessitating trial mixing for quality control. Here seven hardened high-volume fly ash concrete mixtures with different mix proportion with optimized type of AEA and order of adding AEA and a co-polymer superplasticizer (SP) [28,29] were used.

Table 1 presents the composition used in the analysis, with the mix code specifying the water-to-binder ratio (w/b), fly ash-to-binder ratio (FA/b), and whether an air-entraining agent (AEA) was used. For instance, "0.40-33 AEA (ID: 4AE)" indicates a w/b ratio of 0.40, a FA/b ratio of 0.33, and air-entrained concrete. On the other hand, "0.40-33 0 AEA (ID: 4NAE)" represents a code for a non-air-entrained counterpart. The choice of concrete mixes was based on the standard requirements outlined in [33] for exposure class XF4 (involving high saturation, seawater, and de-icing agents). However, some deviations from the standard were made: the fly ash-to-cement ratio was increased to 0.52, exceeding the maximum limit of 0.33 specified by the standard; the silica fume content in the binder was set at 4%, which is below the required minimum of 6%; and the effective water-to-cement ratio was higher than 0.45 [28].

Table 1. Composition of concrete mixes (kg/m^3).

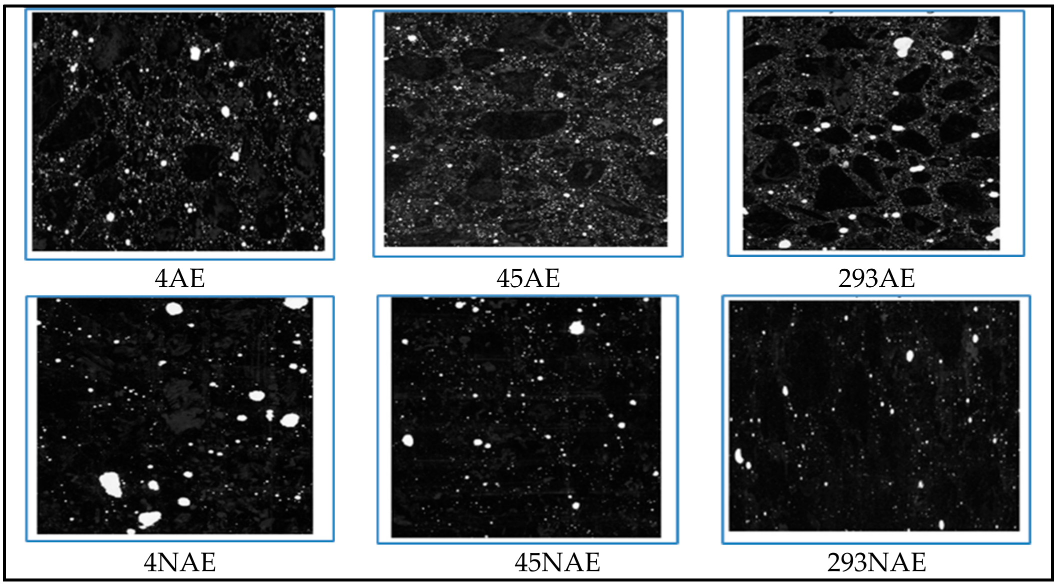
Mix Code	**0.4, 33%FA, AEA**	**0.4, 33%FA, 0 AEA**	**0.45, 33%FA, AEA**	**0.45, 33%FA, 0 AEA**	**0.293, 33%FA, AEA**	**0.293, 33%FA, 0 AEA**	**0.45, 0%FA, AEA**
Constituents Mix ID	**4AE**	**4NAE**	**45AE**	**45NAE**	**293AE**	**293NAE**	**45AE0FA**
Norcem Anleggsement FA	256.2	253.7	240.8	238.1	298.3	293.7	328.8
Elkem Silica Fume	13.9	13.8	13.1	12.9	16.2	16.0	13.7
Norcem Fly Ash	78.5	77.7	73.7	72.9	91.3	89.9	0.0
Free water	139.4	138.0	147.5	145.8	118.9	117.1	154.1
Absorbed water	6.3	6.5	6.3	6.6	6.2	6.7	6.3
Årdal 0/8 mm, A-4956	1062.0	1104.6	1053.6	1104.6	1053.6	1121.5	1053.6
Total moisture (free water)	36.1	34.2	37.9	39.8	37.9	38.1	37.9
Årdal 8/16 mm, A-4751	774.8	805.9	768.6	805.9	768.6	818.2	768.6
Total moisture (free water)	3.9	4.0	4.6	4.8	5.4	5.7	7.7
Mapei Dynamon SX-23	3.0	3.1	2.4	2.8	5.9	6.5	2.7
Incl. liquid part (free water)	2.3	2.4	1.8	2.2	4.6	5.0	2.1
Mapeair 25 1:9	1.5	0.0	1.6	0.0	2.4	0.0	0.7
Incl. liquid part (free water)	1.5	0.0	1.6	0.0	2.4	0.0	0.7
Paste volume, m^3	0.27	0.26	0.27	0.26	0.27	0.26	0.27

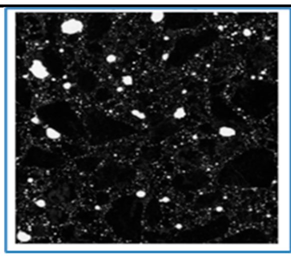
We performed image analysis (IMA) on the polished sections of hardened specimens, each measuring 100 by 100 mm^2, sawn out from a 150 mm cube, following the procedures outlined in [27] and ASTM C457M-16 [10]. Grinding was performed on a Struers

Tegramin-30 using Akasel Aka-Piatto diamond discs with grits of 220, 500, and 1200, applying a consistent pressure of 70-100 N until a reflective surface with well-defined air voids was achieved. The ground surface was then coated three times with a black marker (Edding 850). Air voids were filled with $BaSO_4$ powder (particle size 1–4 μm) using finger tapping and pressing. Excess powder was removed first with a straight-edged plastic ruler and then with a slightly dampened finger. To prevent distortion of air void measurements, cracks and imperfections in the aggregates were painted black under a microscope using a fine-tipped marker. The prepared samples were placed on transparent foil and scanned at 3200 ppi, 16-bit grayscale, using an Epson Perfection V600 Photo scanner. The images were analyzed with a MATLAB script developed by Fonseca [27]. Figure 2 displays the image of each sample used as input for the Python or MATLAB codes.

The performance of the Python code for air void analysis—converted from MATLAB by ChatGPT (Steps 1 and 2, Appendix B) and generated by ChatGPT (Step 3, Appendix C)—was compared with the original MATLAB results to assess both the accuracy of the Python scripts and the capabilities of ChatGPT. Regarding the 3D analysis, the output of one of the samples (4AE) was compared with the 3D analysis of this sample using a Zeiss Metrotom 1500 CT scanner with VGStudio Max 3.0 software for image analysis [15]. It should be noted that stereology (stereology is a scientific method used to quantify three-dimensional (3D) structures based on two-dimensional (2D) images or sections) and tomography (tomography is an imaging technique used to create detailed cross-sectional (slice) images of an object or body by analyzing the data collected from multiple angles) are entirely different; however, they can be comparable in characteristics like the trend of void frequency and spherical diameter.

CT (computed tomography) is an imaging technique that uses X-rays to create detailed cross-sectional images (slices) of an object or body. Multiple 2D images are taken from different angles around the object, and these are reconstructed to form a 3D representation. CT provides direct visual information about the internal structure, allowing for the identification of various tissues, organs, or materials based on their densities. However, stereology is a method used to quantify 3D structures based on 2D images, such as microscope slides. It uses mathematical and statistical techniques to estimate volume, surface area, number, and other characteristics from these 2D sections. While CT offers high-resolution images for direct visual analysis, stereology enables quantitative 3D measurements from 2D sections through statistical methods.





45AE0FA

Figure 2. The scanned image as an input for MATLAB and Python.

3. Results

3.1. Conversion of MATLAB to Python

3.1.1. Converted Code

The void systems in the hardened concrete of the seven mixes in Table 1 (Figure 2) were analyzed using the converted code and combined converted code. To validate the accuracy of our Python code, we conducted a thorough comparison with the results generated by the original MATLAB code. Figure 3 presents this comparative analysis by plotting the values calculated by MATLAB on the x-axis against those calculated by Python on the y-axis for each parameter. A strong correlation was observed, with all three plots demonstrating an R^2 value of 1.00. This correlation indicates that not only does the Python code replicate MATLAB's functionality but it does so with high precision across multiple samples and parameters.

Each of the three air void parameters analyzed—total air void content, specific surface area, and spacing factor—play a critical role in assessing the durability and freeze–thaw resistance of concrete. Total air void content is essential as it indicates the overall volume of air within the concrete, while specific surface area and spacing factor provide insights into the distribution of these voids, which can impact durability. The ability of our Python code to accurately reproduce these parameters underscores its capability to perform robust and comprehensive air void analyses, like MATLAB.

One of the notable benefits of using Python in this study is that it is an open-source alternative to MATLAB, making it more accessible for researchers and practitioners in various settings who may not have access to MATLAB. Python's ecosystem, particularly libraries like OpenCV, NumPy, and SciPy, proved to be efficient in the image-processing tasks essential to this analysis.

Despite the high correlation coefficients observed, it is essential to note the slight computational differences between MATLAB and Python, which may contribute to minor variances in output under certain conditions. These differences are due to intrinsic variations in the image-processing functions and floating-point arithmetic between the two programming environments. However, our analysis showed that these differences are negligible for practical purposes, with the Python results aligning well within the acceptable tolerance limits for air void analysis in concrete. In summary, each tool offers distinct advantages and limitations that impact the effectiveness of data analysis and usability in the field.

1. **Strengths and Limitations of Python**: Python is an open-source language, meaning it is freely available and benefits from a large, collaborative community. Python offers extensive libraries for image processing (like OpenCV and skimage) and data analysis, allowing for flexible and customizable code development. However, Python's flexibility comes with limitations as it requires more effort to configure and integrate compared to MATLAB's built-in functions.
2. **Strengths and Limitations of MATLAB**: MATLAB is widely recognized for its powerful image-processing and data analysis capabilities and is commonly used in engineering fields. However, MATLAB is a commercial software, which can limit

accessibility due to licensing costs. Additionally, MATLAB's proprietary nature may reduce its flexibility for some applications compared to the open-source Python environment.

3. **Comparative Performance in This Study**: When comparing the specific outputs for the air void analysis, both tools provided comparable results in terms of accuracy, as shown by the high correlation coefficients in our analysis. However, in terms of code length and efficiency, Python, especially with the assistance of ChatGPT for code compaction, allowed us to create a streamlined version that was easier to modify for further testing (Section 3.1.2.).
4. **Limitations of Methodologies Employed**: Both Python and MATLAB scripts rely on image-based analysis techniques, which are inherently limited by the resolution and quality of the images used. Any inaccuracies in image capture (e.g., low resolution, poor contrast) can affect the results.

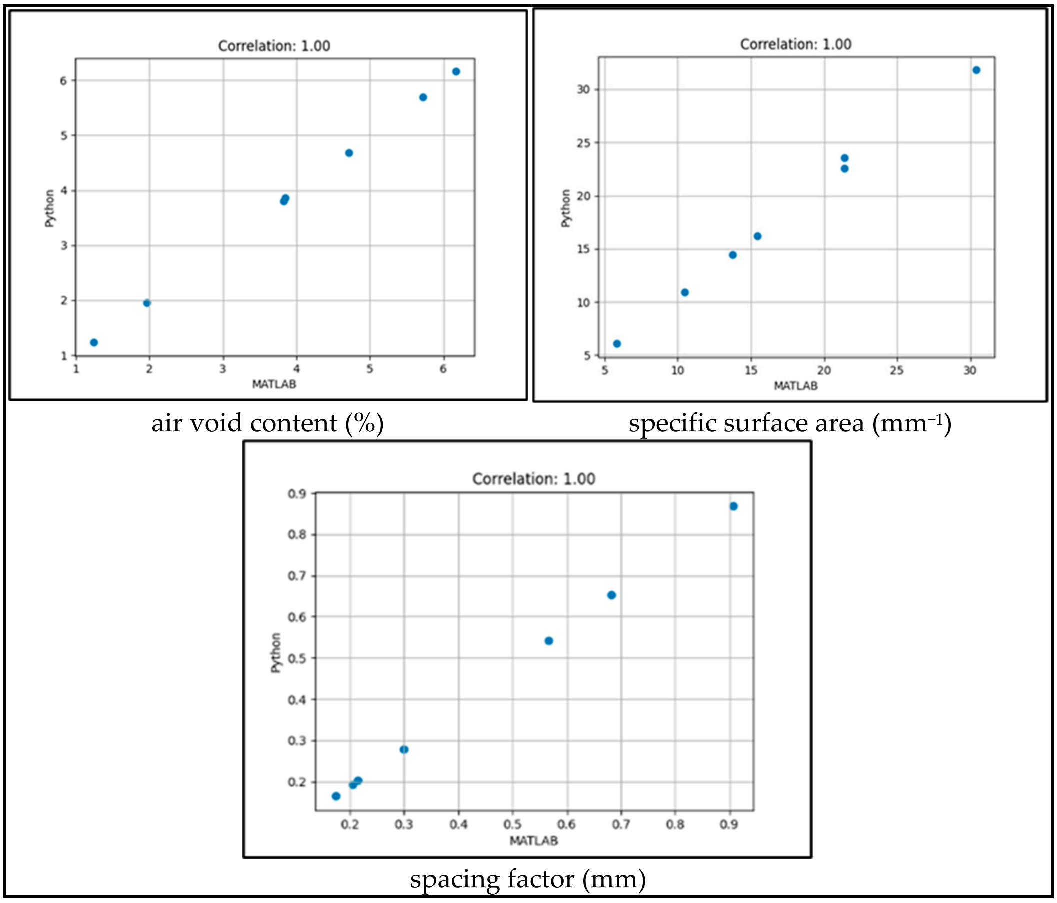


Figure 3. The results of MATLAB vs. Python based on converted code by ChatGPT.

3.1.2. Combined Converted Code

The combined converted code, as presented in Appendix B, merges all three steps of the MATLAB process into a single Python script. It should be noted that the achieved images (Figure 2) were inserted into the codes, and the calculation is based on the binary images (Figure 4). Figure 5 shows the frequency of the number of detected voids, and Figure 6 illustrates the cumulative void volume vs. the diameter. The graphs obtained for both sets of the number of voids and void content are identical to the results reported by MATLAB scripts analyzing the same seven polished sections, as documented in [29]. Hence, ChatGPT was able to translate the well-functioning MATLAB code into a compact Python script and make it work in a similar way, giving the same results with relatively little effort.

In summary, the analysis confirms that the newly developed Python code, supported by ChatGPT in converting the code components, performs air void analysis with a high level of accuracy comparable to the established MATLAB code. This opens new avenues for using Python-based tools in concrete research, offering a cost-effective, flexible, and widely accessible solution for analyzing the air void structure in concrete.

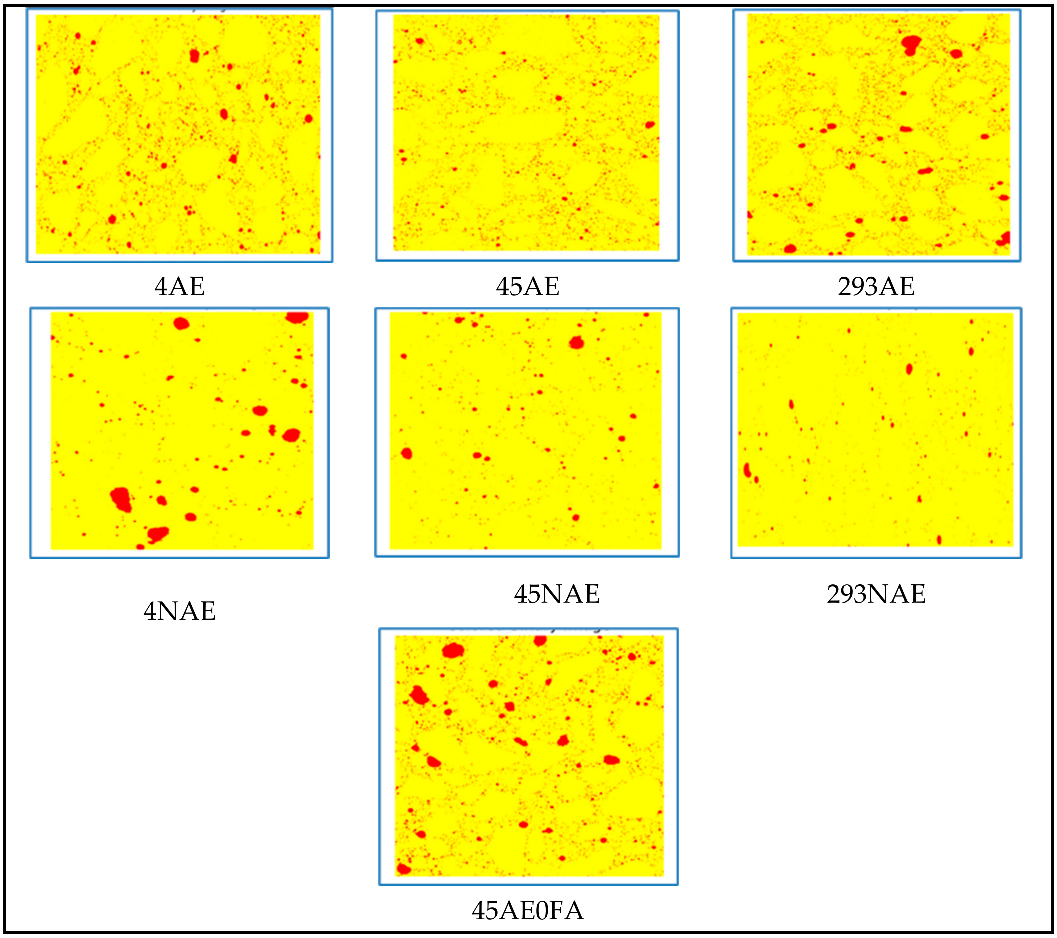


Figure 4. The binary image of samples.

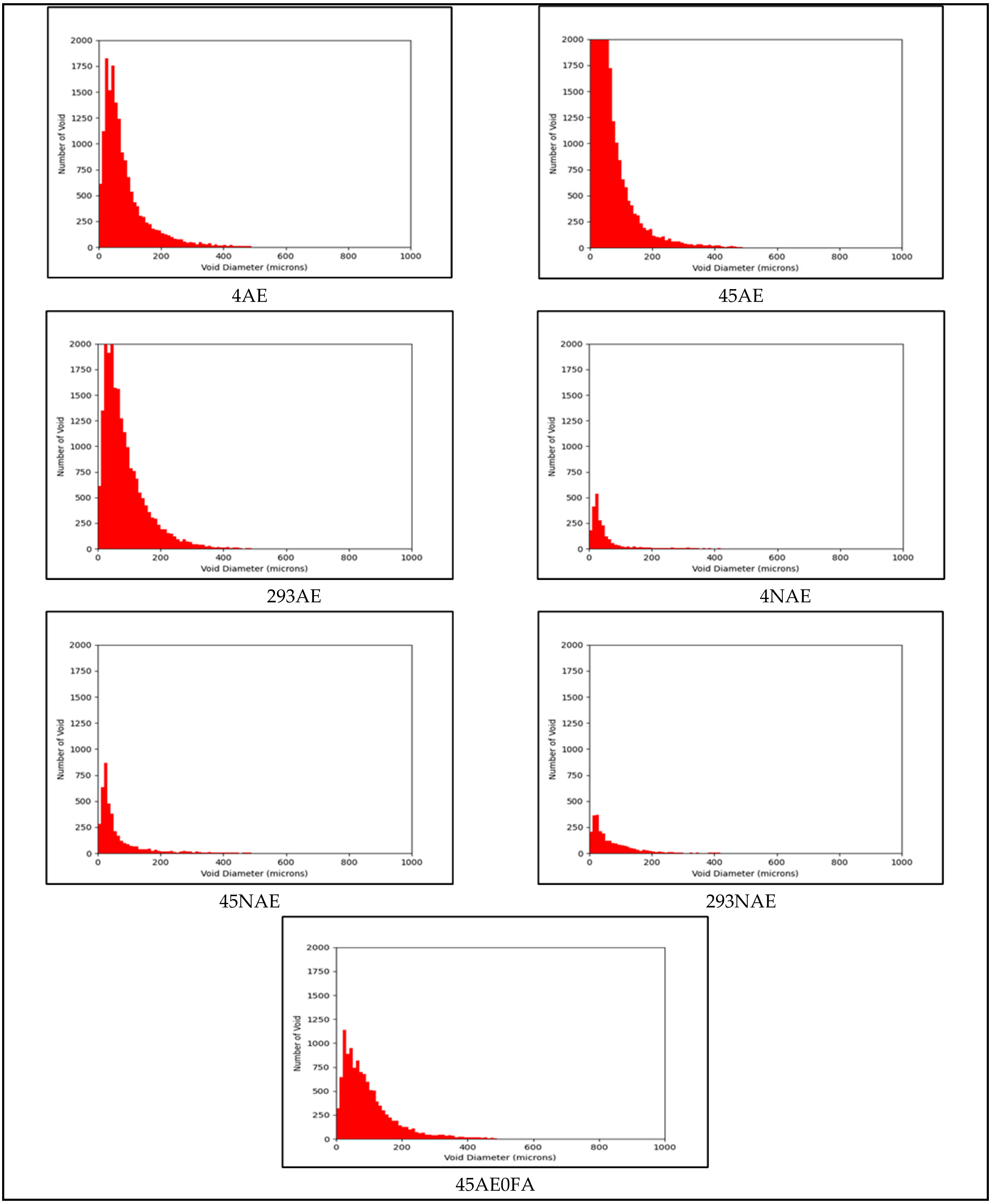


Figure 5. The number of voids vs. void diameter (by 2D polished sections).

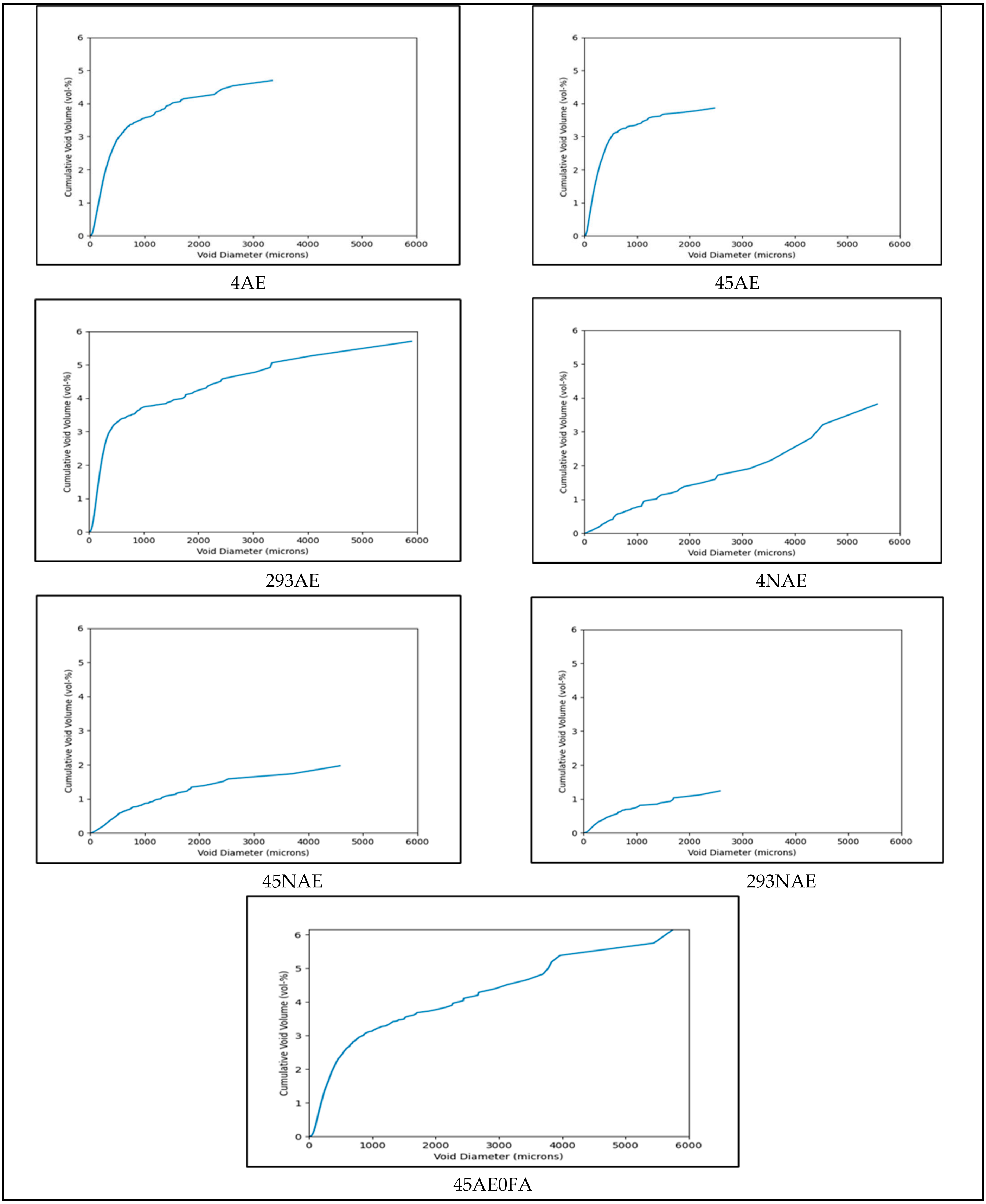


Figure 6. The cumulative void volumes vs. void diameter (by 2D polished sections).

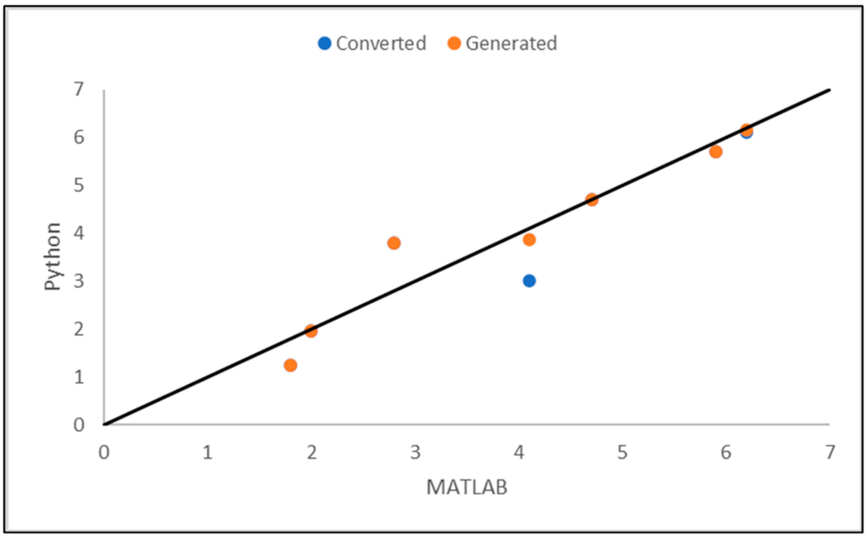
3.2. Generation of Python by Prompt

Figure 7 shows a similar comparison to that in Figure 3 but this time between the air void analysis computed by the original MATLAB code (x-axis) and the air void analysis by the combined converted Python code (Appendix B), as well as the generated code produced without reference to Fonseca and Scherer's MATLAB code (Appendix C). A closer examination reveals some correlation in total air void content between the two, though it is weaker than in Figure 3, with notable discrepancies, especially in specific surface area and Powers' spacing factor, when comparing the ChatGPT-generated Python code to the original MATLAB code. These discrepancies indicate significant errors in the generated code, underscoring the limitations and challenges associated with relying solely on AI-generated solutions for complex computational tasks. Further refinement and validation of the generated code are essential to resolve these discrepancies and ensure reliable, accurate computational results.

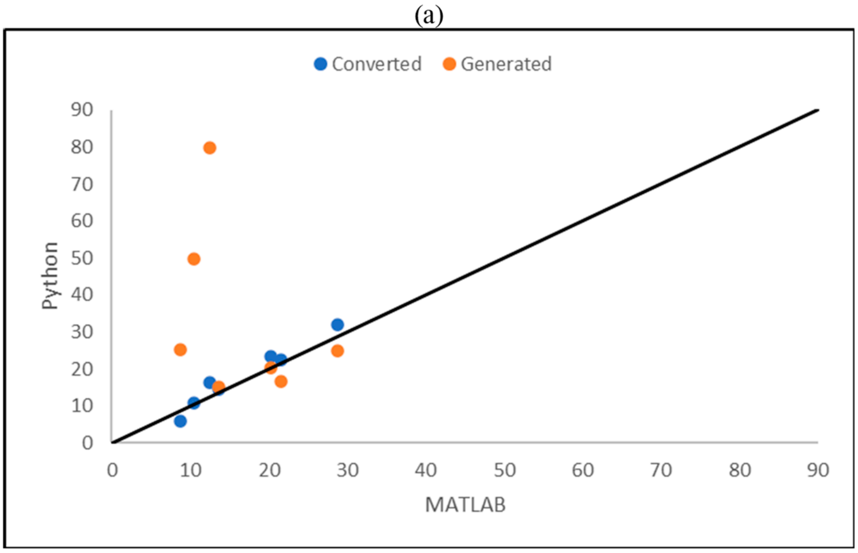
In Table 2, we compare the two codes presented in Appendices B and C. Overall differences in approach and complexity between the two scripts can lead to variations in the computed metrics, especially in specific surface area and Powers' spacing factor, as shown in the comparison. ChatGPT 3.5 requires more detailed prompts to produce results comparable to the MATLAB code of Fonseca and Scherer. Both codes (combined converted and generated) yield similar results for air void content in most samples, suggesting that both methods effectively detect and quantify void spaces within the images. However, for specific surface area and spacing factor, there are noticeable differences, indicating the need for highly detailed prompts based on the ASTM C457 equations.

Table 2. Combined converted code from MATLAB vs. generated code by prompts.

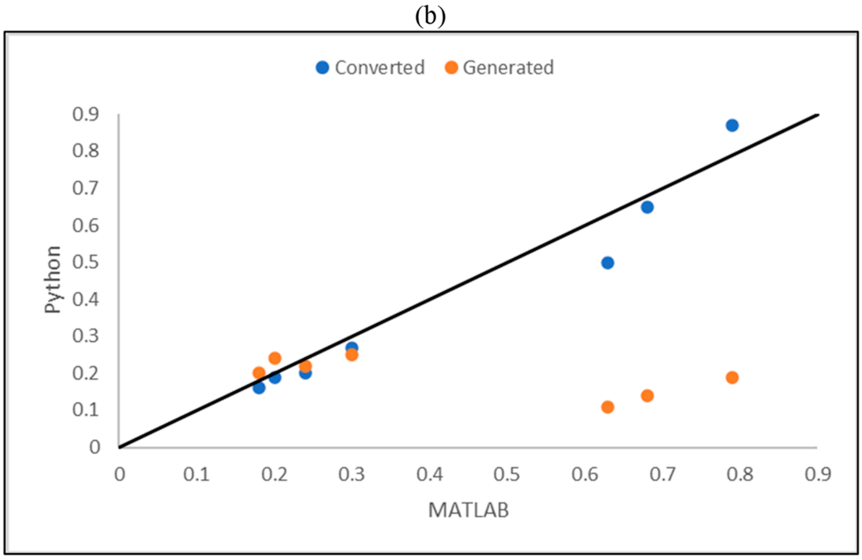
Aspect	Combined Converted	Generated
Library Dependencies	OpenCV, NumPy, scikit-image, Matplotlib	OpenCV, NumPy
Functionality	Comprehensive analysis and visualization	Basic image thresholding and contour analysis
Image Processing	Extensive image analysis including thresholding and measurement	Basic image processing such as thresholding and contour finding
Input Handling	Loads image file, performs optional cropping	
Image Display	Matplotlib (plt) for display	OpenCV (cv2) for displaying thresholded image
Dependency	Uses skimage for Otsu's method	Only relies on OpenCV and NumPy
Thresholding Application	Utilizes Otsu's method for thresholding	
User Interaction	Uses input () for user input	Uses input () and OpenCV GUI for cropping
Output Presentation	Generates histograms and scatter plots for visual representation	Prints metrics related to contours found in the image
Computational Complexity	Higher	Lower



(a)



(b)



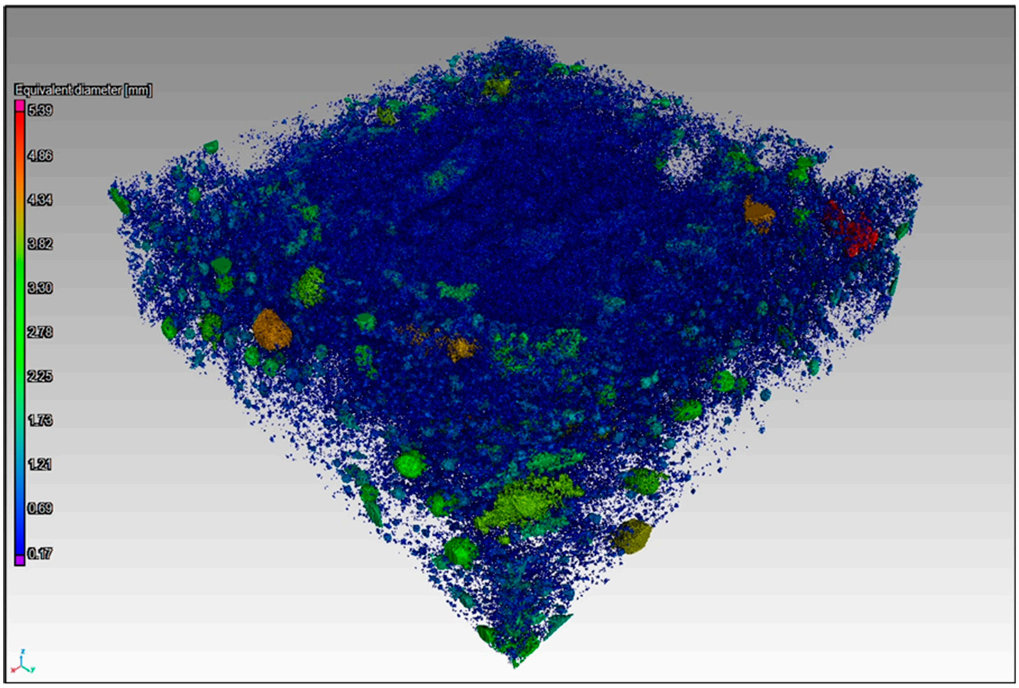
(c)

Figure 7. The results of the air void analysis by MATLAB vs. by Python combined converted and generated by ChatGPT (Appendices A and B): (**a**) air void content (%), (**b**) specific surface area (mm^{-1}), and (**c**) spacing factor (mm).

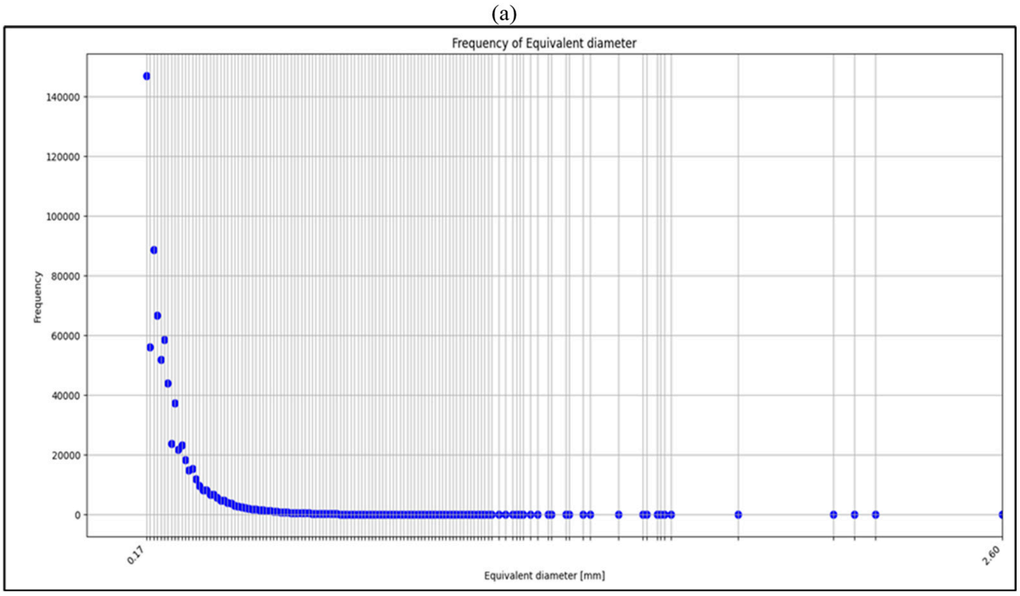
3.3. Computational Tomography vs. Stereology

3.3.1. CT Scanner

In assessing the precision of 3D image reconstruction, a comparison was made between the frequency and diameters of voids derived from the CT scanning results [15]. Figure 8 illustrates the CT scanner output and the frequency of air voids detected based on their equivalent diameter for sample 4AE. A total of 652,809 voids were identified with equivalent diameters ranging from 1 to 300 μm, 90,094 voids were detected with diameters between 300 and 1000 μm, and 1050 voids were found with diameters between 1000 and 2000 μm. Figure 9 also shows the cumulative void volume as a function of void diameter, determined through 3D analysis using the CT scanner. This plot includes one scan of the large concrete slice (20 mm by 100 mm by 100 mm) used for image analysis and nine scans of individual small cubes sawn from the large slice [15]. These results indicate that sample size may influence CT scanner results. In this study, we retained the large sample size because it was also used for image analysis.



(a)



(b)

Figure 8. The equivalent diameter of detected air voids by the CT scanner: (**a**) visualization of all voids in large (100 mm by 100 mm by 20 mm) specimen, (**b**) frequency (sample 4AE).

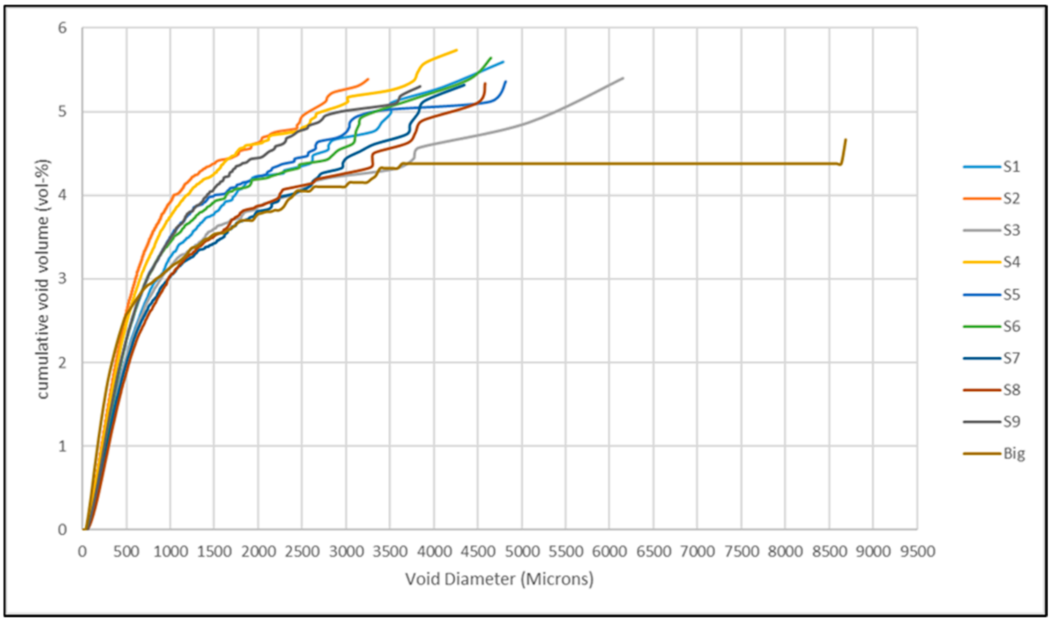


Figure 9. The results of the 3D analysis by the CT scanner (sample 4AE).

Figure 10 shows the sphericity of detected voids by the CT scanner versus the surface area (mm^2) and volume of voids (mm^3). The sphericity of a void measures how closely the shape of the void resembles a perfect sphere. It is defined as the ratio of the surface area of a sphere (with the same volume as the void) to the actual surface area of the void. Mathematically, sphericity (ϕ) is given by [34]

$$\text{Sphericity}(\phi) = \frac{\text{Surface area of a sphere with the same volume as the void}}{\text{Actual surface area of the void}}$$

For a perfect sphere, the sphericity value is 1. As the shape of the void deviates from a sphere (becomes more elongated or irregular), the sphericity decreases, with values ranging between 0 and 1 as follows:

$$\emptyset = \frac{\pi^{\frac{1}{3}}(6V)^{2/3}}{A}$$

where V is the volume of the void, and A is the actual surface area of the void.

It should be noted that the observed surface area in the Figure 10 refers to the total area that covers the entire outer surface of a three-dimensional void, considering all of its contours and dimensions. Hence it accounts for the entire surface of the air voids, including any curves, slopes, or irregularities. However, the projection area (in the XY-plane) is the area of a void when viewed directly from above, onto a flat, two-dimensional plane that corresponds to the XY-plane of the object. Therefore it represents the "shadow" or outline of the air voids as seen from a perpendicular viewpoint to the XY-plane (The size of void which appears in 2D when looking at it from top-down view). The surface area vs XY-plane area of detected voids is shown in Figure 11. The results clearly indicate the differences between these values. As expected the projected area is much smaller than the corresponding void surface, but as seen from Figure 11 it is even smaller than the ratio (area circle/area void) which should be 1/4 for a perfect sphere.

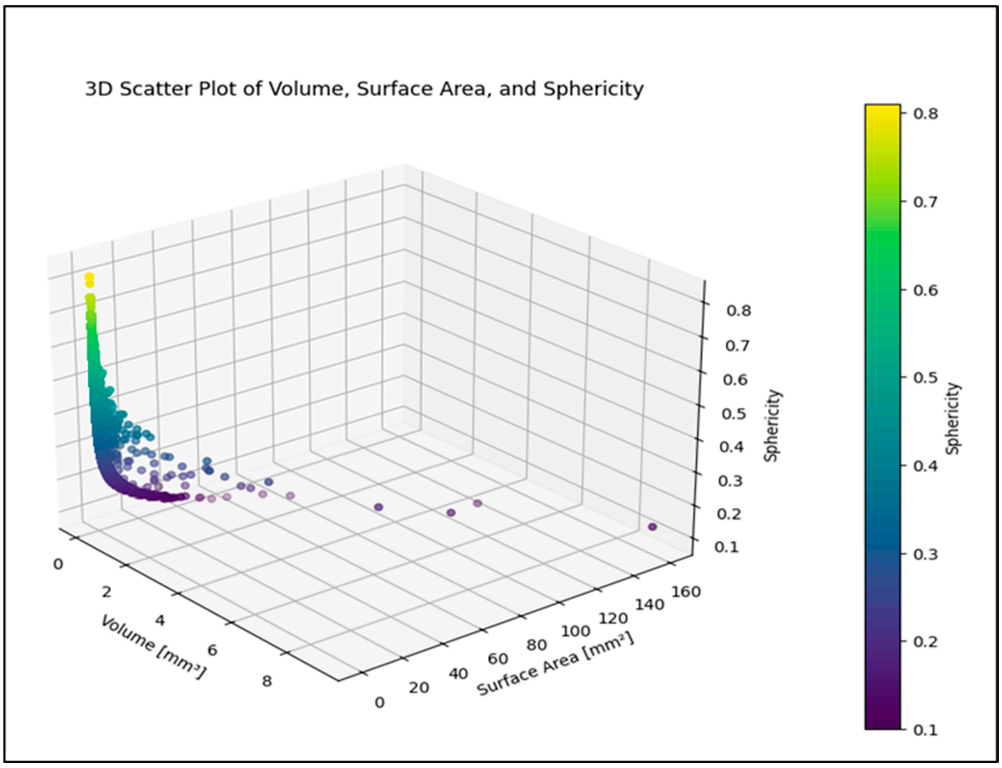


Figure 10. The sphericity of detected voids. (sample 4AE).

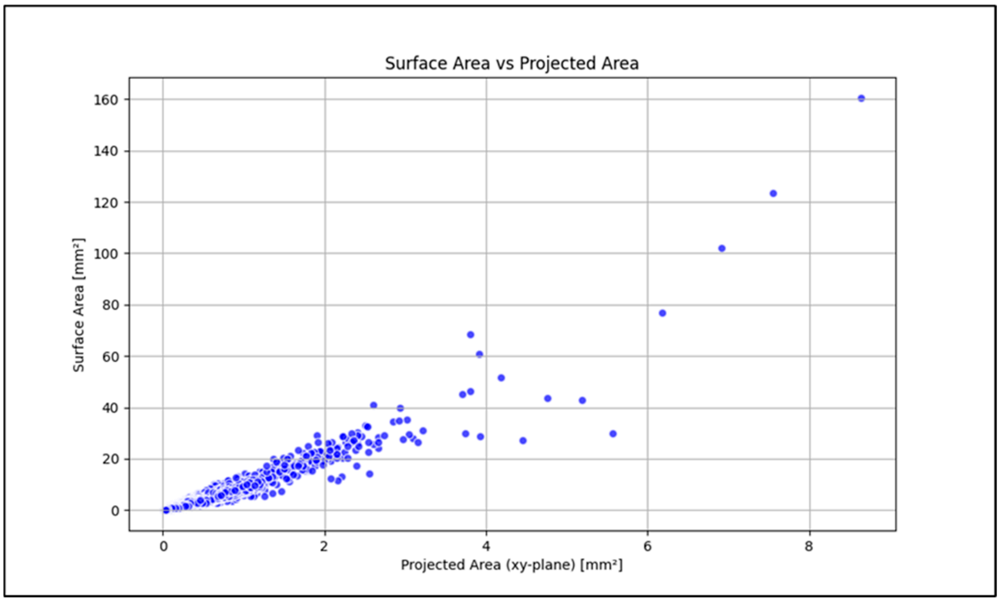


Figure 11. The surface area vs. projected XY area (sample 4AE).

3.3.2. Stereology

The Saltykov method is used in material science to estimate the size distribution of particles (like air voids) in a sample based on the 2D slice of the sample. It is a way to understand the 3D size distribution from a 2D view [27,35]:

$$\mathrm{N_A(i,j) = N_v(j)\Delta\left\{\sqrt{j^2-(i-1)^2}-\sqrt{j^2-i^2}\right\}}$$

where N_A represents the number of profiles per unit area and N_V represents the number of spheres per unit volume, and Δ is the bin which shows the particle classes. A Python script was applied (Appendix C) for fast and accurate calculation. This Python script focused on identifying circular air voids within the binary image and calculating their diameters. After computing these diameters, the code organizes them into a histogram, which shows the distribution of different diameters within the image. This code estimates the radius of the air voids, assuming they have circular shapes (If the shape is not a perfect circle, it gives "equivalent diameter"—the diameter of a circle that has the same area as the non-circular shape.). The area of a circle is given by Area = $\pi \times$ radius^2, so $r = \sqrt{A/\pi}$. N_V is the number of air voids per unit volume for each diameter range (= size distribution by number density). In summary, considering Figure 6 (sample 4AE) and Figure 9, the 2D image analysis has a similar trend as those measured by a CT scanner. However, the number of detected voids by the CT scanner (specially in small samples) is much higher than by stereological analysis of the 2D image by Python script. It can be attributed to the resolution differences of these two methods. The results in Figures 8 and 12 showed, however, that 3D-scanning can give comparable results to reconstructed 2D polished sections (stereological analysis).

As previously discussed (refer to Figure 11), the surface area of a 2D section (for instance, projection XY) can significantly differ from the surface area of a 3D image. To address this discrepancy, we modified the 3D reconstruction from the 2D image using the following approach:

1. The sphericity of all 770,447 voids detected by the CT scanner was categorized based on their XY projection area. The categorization was performed in intervals of 0.01 mm^2 for areas under 0.1 mm^2, 0.1 mm^2 intervals for areas up to 2 mm^2, and 0.5 mm^2 intervals for areas up to 5 mm^2. Python scripts were used for efficient computation (please see Figure 13).
2. The circular surface area of the 2D image was calculated through image analysis by detected diameters.
3. The mean value obtained from the first step was applied as a correction factor (k) for the surface area of the 2D image.
4. The void volumes were then reconstructed using this modified approach. So, we proposed the following equation for the volume of reconstructed voids based on the 2D image:

$$V_{rec} = \frac{1}{6} \cdot (\frac{k \cdot A_{2D}}{\pi^{1/3}})^{3/2}$$

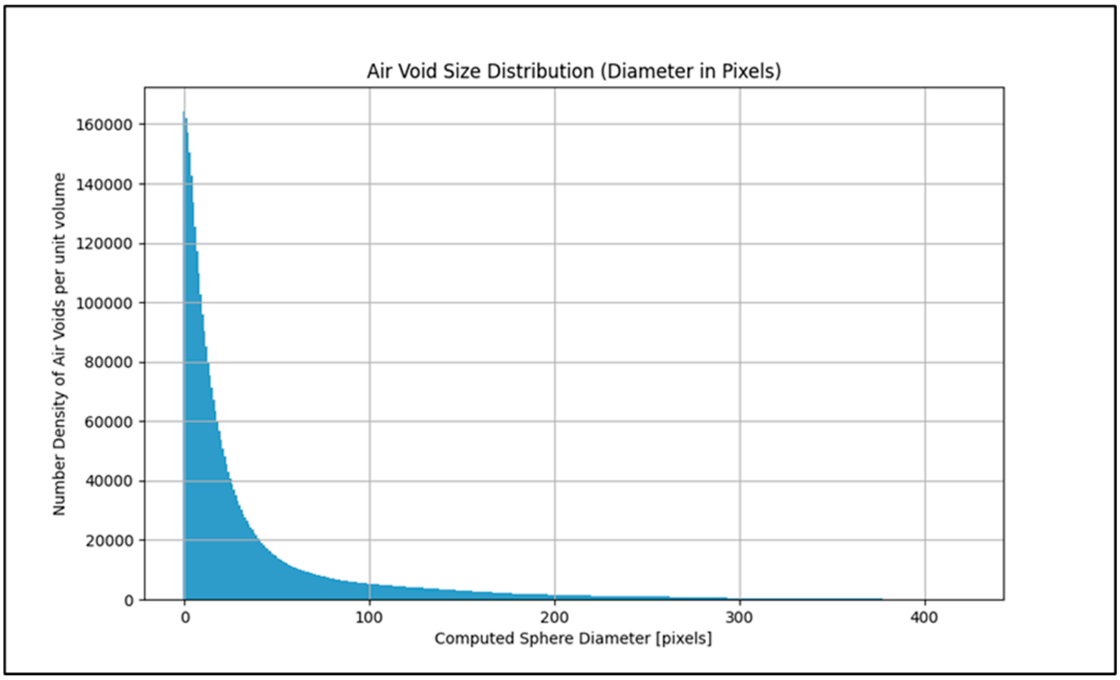


Figure 12. The computed sphere distribution based on Saltykov's classical sphere reconstruction method (sample 4AE).

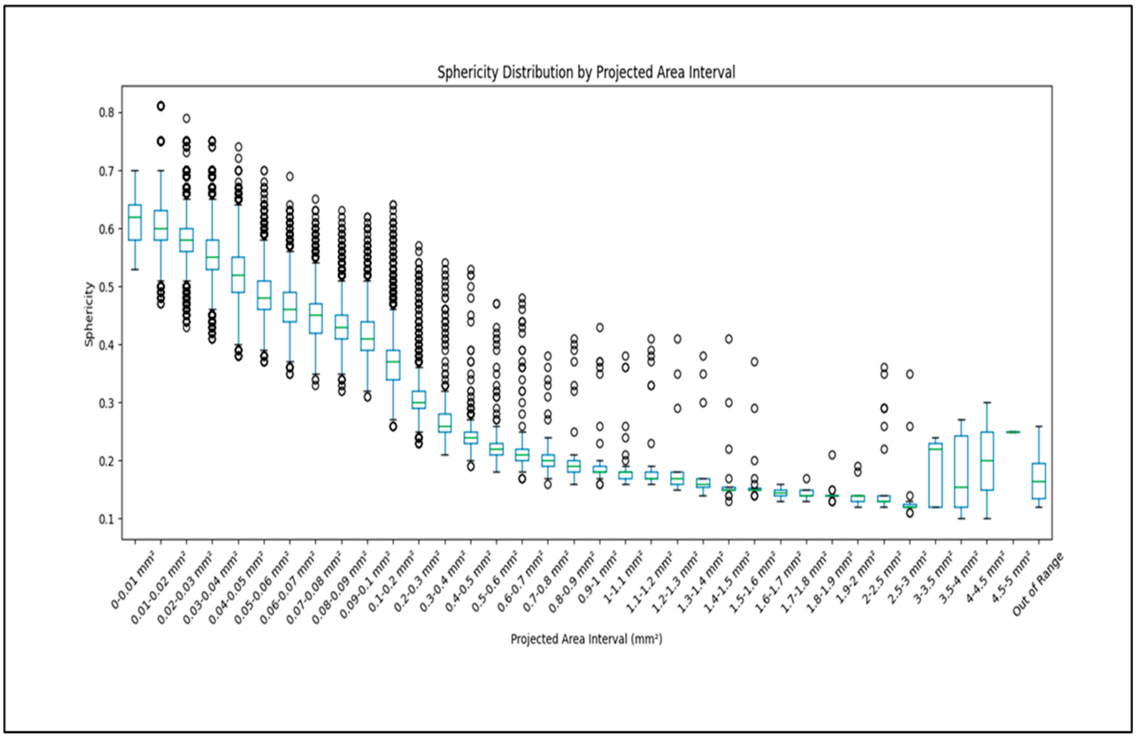


Figure 13. Sphericity distribution (sample 4AE).

The Figure 13 shows that as the projected area of the voids increases, there is a general trend toward lower sphericity values like in Figure 10. This trend suggests that larger voids tend to be less spherical. This could be because larger voids are more likely to have irregular shapes or more complex geometries that deviate from a simple spherical shape. The box plots indicate the median, quartiles, and variability in sphericity for each area range, with the outliers providing further insight into the diversity of void shapes. Figure 14 compares the corrected void volume V-rec to the detected diameter. It shows a clear relationship between the void size (as detected by its diameter) and the reconstructed volume. The blue dots representing the corrected 3D volume, while the red represent the uncorrected 2D area. The deviation between the two data sets indicates the significance of the correction factor k in the equation for reconstructed void volume and the method's efficacy in adjusting the 2D measurements for more accurate 3D volume estimates.

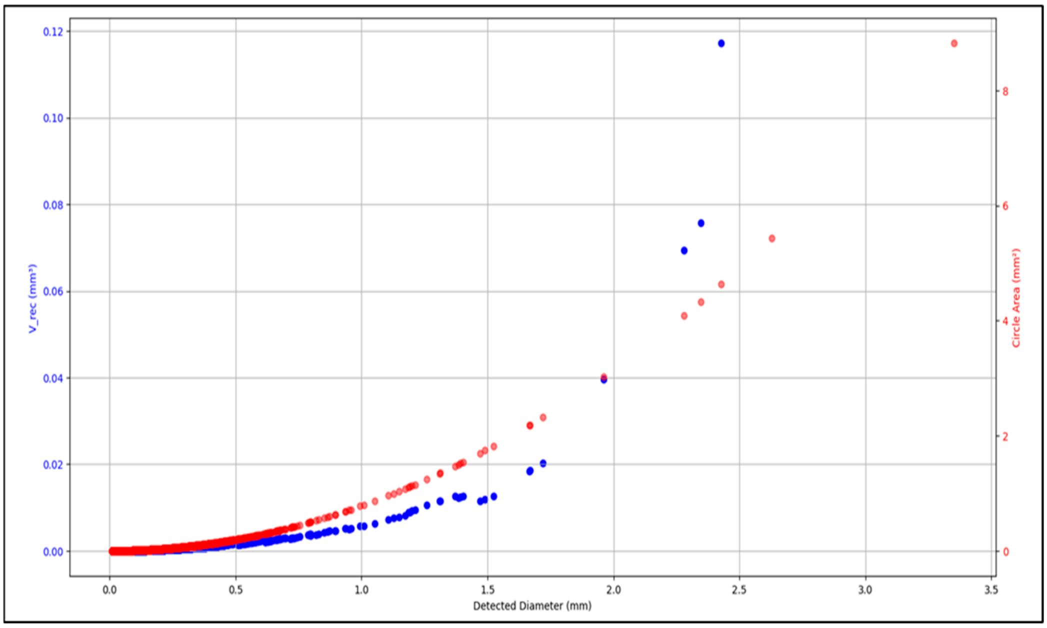


Figure 14. Corrected volume (V-rec) and circle area of 2D voids vs. detected diameter.

4. Conclusions

This study explored the application of Python alongside MATLAB in analyzing air void systems in concrete, emphasizing Python's growing role and potential in this field. By leveraging AI tools like ChatGPT, we demonstrated that Python can effectively streamline the code generation process for air void analysis, offering a practical alternative to MATLAB with comparable results. Our findings suggest that transitioning between programming languages for such analyses can be achieved with minimal effort and high accuracy.

We also examined potential sources of discrepancies between the results generated by MATLAB and Python. Minor differences may arise due to variations in library implementations, numerical precision, or default settings across these platforms, especially in handling image processing and mathematical functions. While these discrepancies were minimal, they underscore the importance of understanding each tool's unique characteristics to ensure consistency in quantitative analysis. Additionally, this discussion highlights the need for further research into Python's accuracy and reliability, especially when applied to complex civil engineering calculations traditionally dominated by MATLAB.

Furthermore, this study extended 2D Python-generated air void parameters to 3D interpretations using CT scan data. While stereology and tomography are distinct methods, the consistency in void frequency and spherical diameter trends indicates that Python, supported by AI, is promising for future applications in 3D void analysis. However, further research is needed to fully realize AI's potential in this domain.

Key insights include the observed reduction in void sphericity with increasing area, and the development of a sphericity correction method to improve 3D volume estimations derived from 2D data. This correction method is especially valuable when 3D imaging resources are unavailable, enhancing the accuracy of material property assessments.

In conclusion, integrating Python, aided by AI tools like ChatGPT, offers significant advantages in the advanced analysis of concrete air voids. Python's accessibility, flexibility, and cost-effectiveness, combined with its robust AI and machine learning capabilities, make it an increasingly attractive option for civil engineering applications. As Python continues to expand in academic and professional contexts, it has the potential to transform air void analysis and other complex material studies, making advanced analyses more accessible and adaptable to a wide range of research and industry needs.

Author Contributions: Conceptualization, I.A. and S.J.; methodology, I.A. and S.J.; software, I.A.; formal analysis, I.A.; investigation, A.S.; data curation, A.S.; writing—original draft preparation, I.A.; writing—review and editing, S.J. and A.S.; visualization, I.A.; supervision, S.J. All authors have read and agreed to the published version of the manuscript.

Funding: Open Access Funding by the University for Continuing Education Krems.

Data Availability Statement: Data will be made available on request.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

Prompt for Code Development Using ChatGPT

- First Step: Code Conversion
  - ***Authors***> Please transfer the following MATLAB codes to one script of Python. (We pasted the MATLAB scripts, separately)
    - ✓ ***ChatGPT***> Here's the Python code equivalent of the given MATLAB code: (The Python code was generated in just 2 or 3 s, and it worked correctly, see Figure 2)

  This process continued for the two other scripts of MATLAB. Then,

- Second Step: Combined Code Conversion
  - ***Authors***> Please combine all these three codes into one compacted script.
    - ✓ ***ChatGPT***> Certainly! Here's the compacted script that combines all three Python code snippets. (However, the generated code was wrong)

So, we returned to the chat (before pasting the reconstruct MATLAB script) and asked it to combine the first and second scripts. The generated code was missing parts and contained errors.

- ***Authors***> The paste solids and threshold should be asked before running.
  - ✓ ***ChatGPT***> Certainly! Here's the updated code that asks for the paste solids and threshold before running the basic analysis function. (Then, the code was generated, and we continued the chat to add or ignore minor issues)

For the final steps of 2D analysis and giving the results of the void system, we asked it to complete the code by generating some graphs.

- ***Authors***> We need to graph the frequency of voids and cumulative voids vs. the void size.
  - ✓ ***ChatGPT***> Certainly! Here's an updated version of the code that includes plotting the frequency of voids and cumulative voids versus void size.

The code was generated and worked. We just continued chatting about minor issues like graph color or axis captions. As mentioned earlier, we further asked it to complete the generated code for 3D analysis. Before fulfilling this step, we tried to check the quality of the code through the following questions:

- **Authors**> What is this code? (And the Python code was pasted)
  - ✓ ***ChatGPT***> This code appears to be a Python script for analyzing an image. Here is a breakdown of its functionality: The code imports necessary libraries such as OpenCV (cv2), NumPy (np), scikit-image filters (skimage. filters), scikit-image measure (skimage. measure), os, PIL (Image), and matplotlib—pyplot as plt. The function read_image takes a filename, folder path, cropping flag, and threshold value as input. It reads the image using OpenCV (cv2.imread) and converts it to grayscale. If the cropping flag is set, the user can select a region of interest (ROI) in the image. The function returns the file path, original image, and grayscale image. The function analyze_image takes the file path, original image, grayscale image, paste/solids ratio, and custom threshold value as input. It analyzes the image by performing the following steps: Retrieving the image dimensions and resolution from the

image file using the PIL library calculating the resolution and size of each pixel in microns. If a threshold value is not provided, it uses Otsu's method to find the threshold value for image binarization. Creating a binary image based on the threshold value. Labeling the connected components (regions) in the binary image. Measuring properties of the connected components using region props. Extracting relevant properties such as the components area, diameter, and radius. Calculating the void content as a percentage of the image area. Calculating the specific surface (surface area per unit volume). Computing the Powers spacing factor based on the paste/solid ratio and void content. Displaying the grayscale image using OpenCV. Printing the number of detected voids. Sorting the diameters and areas of voids in ascending order. Calculating the cumulative void content. Calculating the number of voids for different diameter ranges. Plotting the number of voids for different diameter ranges as a bar chart. Plotting the void diameters vs. cumulative void content. The main part of the code checks if the script is being run directly (not imported as a module) by using the __name__ variable. It calls the read_image function to read and crop the image, sets the paste/solids ratio and threshold values, and then calls the analyze_image function to analyze the image. Overall, this script reads an image, performs image analysis, and generates plots to visualize the void content and void distribution based on the provided parameters.

- ***Authors***> Update the code for 3D stereological analysis and show the 2D graph for spherical diameter vs. the number of voids.
  - ✓ ***ChatGPT***> To update the code for 3D stereological analysis and show a 2D graph for spherical diameter vs. the number of voids, you will need to make several modifications. Here's an updated version of the code. (And it prepared the script)
- ***Authors***> It is the cumulative graph. We need the frequency graph to show the number of voids regarding its spherical diameter.
  - ✓ ***ChatGPT***> Apologies for the confusion. To create a frequency graph showing the number of voids based on their spherical diameter, you'll need to modify the code as follows. (The code was generated entirely, and we stopped at this point. Please see Appendix B).

➢ Third Step: Code Generation without reference to Fonseca and Scherer's MATLAB code:

In the third step, the scope was to find out the capability of ChatGPT in code generation without the MATLAB script, we explained to it our request as follows:

- ***Authors***> Write a Python script that converts an original grayscale image with pixels going from 0 = black to 255 = white into a black-and-white image. Use Otsu's method for thresholding so that each pixel in the converted image is either 0 = black or 1 = white. Determine the threshold value between black and white in the original grayscale image between 0 and 255 and also enable the option for the user of the code to select and crop the area of the original image that should be analyzed or continue without selecting and crop the area of the original image. Then, for the black and white image, I need to know the air void content A of the sample (%). Then, calculate the specific surface of the air voids, and calculate spacing factor, based on the methods and equation in Procedure C of ASTM C457/C457M-16. Please give the option to crop or not crop the image.
  - ✓ ***ChatGPT***> Here's a Python script that accomplishes your requirements using OpenCV for image processing and calculations: (Appendix C)

Appendix B

Python code converted (based on MATLAB and accumulated void volume plus 3D stereological parameters).

```
# Importing required libraries
import cv2  # OpenCV library for image processing
import numpy as np  # NumPy for numerical operations
import skimage.filters as filters  # skimage.filters for image thresholding methods
import skimage.measure as measure  # skimage.measure for region measurements (e.g.,
area, perimeter)
import os  # os module to handle file paths
from PIL import Image  # PIL for image handling and metadata extraction
import matplotlib.pyplot as plt  # Matplotlib for plotting data and visualizations
import sys  # sys library, though not used here, generally useful for system operations
# Set to None to allow loading very large images
Image.MAX_IMAGE_PIXELS = None  # Removes pixel limit to handle large images
# Function to read an image, crop if needed, and convert to grayscale
def read_image(filename, folder='', crop=False, paste_solids=0.5, thresh=None):
    # Combine folder and filename into a full file path if a folder is specified
    filepath = os.path.join(folder, filename) if folder else filename
    # Load the image from the specified file path
    img = cv2.imread(filepath)
    # If cropping is enabled, allow the user to select a region of interest (ROI)
    if crop:
        roi = cv2.selectROI(img)  # Select ROI on the image
        # Crop the image to the selected ROI area
        img = img[int(roi[1]):int(roi[1] + roi[3]), int(roi[0]):int(roi[0] + roi[2])]
    # Convert the image to grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Return the file path, original image, and grayscale image
    return filepath, img, gray
# Function to analyze the voids in an image
def analyze_image(filepath, img, gray, paste_solids, thresh=None):
    # Define millimeters per inch constant for DPI conversion
    mmpi = 25.4
    # Open the image using PIL to access DPI information
    with Image.open(filepath) as img:
        xR, yR = img.info['dpi']  # Get horizontal and vertical DPI
    # Calculate pixels per millimeter (ppmm) based on DPI
    ppmm = xR.numerator / xR.denominator / mmpi
    # Calculate pixel size in microns
    pixsize = 25400 / xR.numerator
    # Print resolution and pixel size details
    print(f"The resolution of the image is {xR} pixels/inch or {ppmm} pixels/mm.")
    print(f"The size of each pixel is {pixsize} microns.")
    # If no threshold is provided, use Otsu's method to determine threshold
```

```
    if thresh is None:
        threshold_value = filters.threshold_otsu(gray)
    else:
        threshold_value = thresh  # Use custom threshold if provided
    # Create a binary image by thresholding the grayscale image
    binary_image = gray > threshold_value
    # Label the connected regions (voids) in the binary image
    label_image = measure.label(binary_image)
    # Get properties (e.g., area, perimeter) for each labeled region
    props = measure.regionprops(label_image)
    # Calculate the area (in pixels) for each void
    area_pixels = [prop.area for prop in props]
    # Calculate the diameter (in microns) for each void based on area
    diameter_microns = [2 * np.sqrt(prop.area / np.pi) * 1000 / ppmm for prop in props]
    # Convert diameter from microns to millimeters
    diameter_mm = [d / 1000 for d in diameter_microns]
    # Get image dimensions (height and width)
    m, n = gray.shape
    # Calculate the percentage of the image area that is voids
    void_content = sum(area_pixels) / (m * n) * 100
    # Print void content as a percentage
    print(f"The void content is {void_content} percent.")
    # Calculate the total perimeter of all voids
    Ptot = sum([prop.perimeter for prop in props])
    # Calculate the total area of all voids
    Atot = sum([prop.area for prop in props])
    # Calculate specific surface in pixels (surface area per unit volume)
    specsurfpix = Ptot / Atot * 4 / np.pi
    # Convert specific surface to mm^-1 by scaling with ppmm
    specsurf = specsurfpix * ppmm
    # Print the specific surface value
    print(f"The specific surface is {specsurf} mm^-1.")
    # Calculate p_A value based on paste solids and void content
    p_A = paste_solids * (100 - void_content) / void_content
    # Calculate the Powers spacing factor based on specific surface and p_A
    if p_A < 4.342:
        Pspacef = p_A / specsurf
    else:
        Pspacef = (3 / specsurf) * (1.4 * (1 + p_A) ** (1 / 3) - 1)
    # Print the Powers spacing factor
    print(f"The Powers spacing factor is {Pspacef} mm.")
    plt.figure(figsize=(8, 6)) # Plot various analyses of the image and voids
```

```
    # Display the grayscale image
    plt.subplot(2, 2, 1)
    plt.imshow(gray, cmap='gray')
    plt.axis('off')  # Turn off axis
    plt.title('Grayscale Image')
    # Plot a histogram of void diameters
    plt.subplot(2, 2, 2)
    plt.hist(diameter_microns, bins=np.arange(0, 500, 10), color='red')
    plt.xlim(0, 2000)
    plt.xlabel('Void Diameter (microns)')
    plt.ylabel('Number of Voids')
    plt.title('Void Diameter vs Number of Voids')
    # Plot cumulative void content as a function of void diameter
    plt.subplot(2, 2, 3)
    diameters_sorted, areas_sorted = zip(*sorted(zip(diameter_microns, area_pixels)))
    cumulative_void_content = np.cumsum(areas_sorted) / (m * n) * 100
    plt.plot(diameters_sorted, cumulative_void_content)
    plt.xlim(0, 6000)
    plt.xlabel('Void Diameter (microns)')
    plt.ylabel('Cumulative Void Content (%)')
    plt.title('Cumulative Void Content vs Void Diameter')
    # Scatter plot of spherical diameter vs number of voids
    plt.subplot(2, 2, 4)
    num_voids = len(area_pixels)
    plt.scatter(diameter_mm, range(num_voids), color='blue')
    plt.xlabel('Spherical Diameter (mm)')
    plt.ylabel('Number of Detected Voids')
    plt.title('Spherical Diameter vs Number of Detected Voids')
    # Adjust layout to avoid overlapping
    plt.tight_layout()
    plt.show()
if __name__ == '__main__': # Main block to execute the analysis
    # Read the image file and obtain grayscale version
    filepath, img, gray = read_image('4AE.tif', crop=False)
        paste_solids = 0.266 # Set paste solids content
    custom_threshold = None     # Set custom threshold (None uses default threshold
method)
    analyze_image(filepath, img, gray, paste_solids, custom_threshold)    # Analyze the
image using the defined function
```

Appendix C. Python Code Generated from Prompt Only

```
import cv2
import numpy as np
import matplotlib.pyplot as plt  # For displaying images with Matplotlib
# Define the function for Otsu's thresholding
def otsu_threshold(img):
    # Apply Otsu's thresholding to binarize the image
    _, thresh = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    return thresh
# Define the function to calculate air void content
def get_air_void_content(img):
    # Count white pixels (voids) in the thresholded image
    white_pixels = np.count_nonzero(img == 255)
    total_pixels = img.shape[0] * img.shape[1]
    air_void_content = (white_pixels / total_pixels) * 100
    return air_void_content
# Define the function to calculate contour properties
def get_contours_properties(thresh):
    # Find contours in the thresholded image
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    total_area = 0
    total_length = 0
    for contour in contours:
        total_area += cv2.contourArea(contour)  # Sum of areas of contours
        total_length += cv2.arcLength(contour, True)  # Sum of perimeter lengths
    # Calculate the Powers spacing factor and specific surface area
    power_spacing_factor = total_length / total_area
    specific_surface_area = total_area / np.count_nonzero(thresh == 255)
    return power_spacing_factor, specific_surface_area
# Main function to process the image
def main():
    # Prompt user to enter the path to the image
    image_path = input("Enter the path to the image: ")
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)  # Load image in grayscale
    crop_option = input("Would you like to crop the image? (yes/no): ").lower()
    if crop_option == 'yes':
        print("Please select the area to crop. Press Enter to confirm the selection.")
        roi = cv2.selectROI("Select ROI", img)  # Allow user to select region of
interest
        cv2.destroyAllWindows()
        img = img[int(roi[1]):int(roi[1] + roi[3]), int(roi[0]):int(roi[0] + roi[2])]
# Crop the image to ROI
```

```
    # Apply Otsu's threshold to binarize the image
    thresh = otsu_threshold(img)
    # Display thresholded image using Matplotlib
    plt.imshow(thresh, cmap='gray')
    plt.title("Thresholded Image")
    plt.axis('off')
    plt.show()
    # Calculate air void content and contour properties
    air_void_content = get_air_void_content(thresh)
    power_spacing_factor, specific_surface_area = get_contours_properties(thresh)
    # Print results
    print(f"Air Void Content (%): {air_void_content:.2f}")
    print(f"Power Spacing Factor (L): {power_spacing_factor:.2f} mm")
    print(f"Specific Surface Area: {specific_surface_area:.2f} mm^-1")
# Ensure main function runs if script is executed
if __name__ == "__main__":
    main()
```

Appendix D. Python Script Developed for Air Void Distribution

```
# Importing necessary libraries
import cv2  # OpenCV library for image processing
import numpy as np  # NumPy for numerical operations
import matplotlib.pyplot as plt  # Matplotlib for plotting
from scipy.ndimage import label  # Scipy for labeling connected components
# Function to compute a histogram of void diameters in a binary image
def compute_2d_diameter_histogram(image, bin_width=1):
    # Label each connected component (void) in the binary image
    labeled_array, num_features = label(
        image)  # labeled_array assigns a unique number to each component, num_features
gives the count of components
    diameters = []  # List to store calculated diameters of each void
    # Iterate through each labeled component (void)
    for i in range(1, num_features + 1):
        # Create a binary mask for the current component
        component = (labeled_array == i).astype(np.uint8)  # Binary mask of the
component
        # Calculate the area (number of pixels) of the component
        area = np.sum(component)  # Area in terms of pixels
        # Calculate the radius and then the diameter, assuming a circular shape
        radius = np.sqrt(area / np.pi)  # Radius from area using formula for a circle
        diameter = 2 * radius  # Diameter as twice the radius
        diameters.append(diameter)  # Add diameter to the list
```

```
    # Compute histogram of diameters, using bins based on the specified bin_width
    hist, bin_edges = np.histogram(diameters, bins=np.arange(0, max(diameters) +
bin_width, bin_width))
    # Return histogram values and bin edges (excluding last edge, as hist aligns with
bin centers)
    return hist, bin_edges[:-1]
# Function to calculate number density per unit volume (Nv) based on the histogram
def calculate_nv(hist, bin_centers, bin_width):
    num_classes = len(bin_centers)  # Number of diameter classes (bins)
    nv = np.zeros_like(hist, dtype=float)  # Initialize Nv array with zeros, same size
as hist
    # Loop through each bin (diameter class)
    for j in range(num_classes):
        sum_term = 0  # Sum term for calculating Nv for the j-th bin
        # Inner loop accumulates the terms for bins greater than or equal to the
current j-th bin
        for i in range(j, num_classes):
            # Term is the square root of the difference in diameter squared
            term = np.sqrt((bin_centers[i] ** 2 - bin_centers[j] ** 2))
            sum_term += hist[i] * term  # Accumulate weighted count of voids
        # Calculate Nv for the j-th bin by dividing sum_term by bin_width
        nv[j] = sum_term / bin_width
    return nv  # Return the array of Nv values
# Request user to input the path to the image file
image_path = input("Enter the path to the 2D image (e.g., '4AE.tif'): ")
# Read the image in grayscale mode
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
# Apply a binary threshold to create a binary (black and white) image
_, binary_img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)  # Threshold value is
127, creating a binary mask
# Define bin width for the histogram (in pixels)
bin_width = 1  # Unit in pixels
# Compute the 2D diameter histogram of air voids in the binary image
hist, bin_centers = compute_2d_diameter_histogram(binary_img, bin_width)
# Calculate number density (Nv) for each diameter bin
nv_result = calculate_nv(hist, bin_centers, bin_width)
# Plot the results
plt.figure(figsize=(10, 6))  # Create a figure with specified size
# Plot the Nv values as a bar chart with bin_centers as x-axis (diameter)
plt.bar(bin_centers, nv_result, width=bin_width, align='center', alpha=0.7)
plt.xlabel('Computed Sphere Diameter [pixels]')  # Label for x-axis
plt.ylabel('Number Density of Air Voids per unit volume')  # Label for y-axis
```

```
plt.title('Air Void Size Distribution (Diameter in Pixels)')  # Title of the plot
plt.grid(True)  # Enable grid for easier viewing
plt.show()  # Display the plot
```

References

1. Das, A.; Song, Y.; Mantellato, S.; Wangler, T.; Lange, D.A.; Flatt, R.J. Effect of processing on the air void system of 3D printed concrete. *Cem. Concr. Res.* **2022**, *156*, 106789.
2. Wong, H.; Pappas, A.; Zimmerman, R.; Buenfeld, N. Effect of entrained air voids on the microstructure and mass transport properties of concrete. *Cem. Concr. Res.* **2011**, *41*, 1067–1077.
3. Rath, S.; Puthipad, N.; Attachaiyawuth, A.; Ouchi, M. Critical size of entrained air to stability of air volume in mortar of self-compacting concrete at fresh stage. *J. Adv. Concr. Technol.* **2017**, *15*, 29–37.
4. Shah, H.A.; Yuan, Q.; Zuo, S. Air entrainment in fresh concrete and its effects on hardened concrete-a review. *Constr. Build. Mater.* **2021**, *274*, 121835.
5. Asadi, I.; Skjølsvold, O.; Kanstad, T.; Jacobsen, S. Frost-salt Testing Non-air Entrained High-performance Fly-ash Concrete Part II: Parameter Study of Effect of Internal Cracking on Glue Spall Stress. *Nord. Concr. Res.* **2024**, *70*, 43–55.
6. True, G.F. Development of Image Analysis Techniques to Assist Evaluation of Both Air Void Structure and Aggregate Shape Factors in Concrete. Ph.D. Thesis, University of Wolverhampton, Wolverhampton, UK, 2011.
7. Powers, T.C.; Willis, T. The air requirement of frost-resistant concrete. *Highw. Res. Board Proc.* **1949**, *29*, 184–211.
8. Chung, S.-Y.; Sikora, P.; Rucinska, T.; Stephan, D.; Abd Elrahman, M. Comparison of the pore size distributions of concretes with different air-entraining admixture dosages using 2D and 3D imaging approaches. *Mater. Charact.* **2020**, *162*, 110182.
9. Cai, H.; Liu, X. Freeze-thaw durability of concrete: Ice formation process in pores. *Cem. Concr. Res.* **1998**, *28*, 1281–1287.
10. *ASTM C 457-90*; Standard Test Method for Microscopical Determination of Parameters of the Air-Void System in Hardened Concrete. ASTM International: West Conshohocken, PA, USA, 1998.
11. *EN 480-11*; Determination of Air Void Characteristics in Hardened Concrete. European Committee for Standardization: Brussels, Belgium, 2005.
12. Pleau, R. *La Caractérisation du Réseau de Bulles D'air Dans le Béton Durci Comme Outil D'évaluation de la Durabilité au Gel du Béton*; Université Laval: Quebec City, QC, Canada, 1992.
13. Snyder, K.A.; Hover, K.C.; Natesaiyer, K. The Stereological and Statistical Properties of Entrained Voids in Concrete: A Mathematical Basis for Air Void System Characterization. In *Materials Science of Concrete VI*; Wiley: Hoboken, NJ, USA, 2001.
14. Murotani, T.; Igarashi, S.; Koto, H. Distribution analysis and modeling of air voids in concrete as spatial point processes. *Cem. Concr. Res.* **2019**, *115*, 124–132.
15. Asadi, I.; Endrerud, P.E.; Jacobsen, S. X-ray tomography to characterize the air-void system of hardened concrete with varying sample size. In Proceedings of the fib International Congress, Oslo, Norway, 12–16 June 2022.
16. OpenAI. ChatGPT: Optimizing Language Models for Dialogue. Available online: https://chatgpt.r4wand.eu.org/ (accessed on 4 February 2023).
17. Quick uptake of ChatGPT, and more—This week's best science graphics. *Nature*, 28 February 2023.
18. Biswas, S.S. Role of chat gpt in public health. *Ann. Biomed. Eng.* **2023**, *51*, 868–869.
19. Adeshola, I.; Adepoju, A.P. The opportunities and challenges of ChatGPT in education. *Interact. Learn. Environ.* **2023**, 1–14.
20. Biswas, S.S. Potential use of chat gpt in global warming. *Ann. Biomed. Eng.* **2023**, *51*, 1126–1127.
21. Cheng, S.W.; Chang, C.W.; Chang, W.J.; Wang, H.W.; Liang, C.S.; Kishimoto, T.; Chang, J.P.C.; Kuo, J.S.; Su, K.P. The now and future of ChatGPT and GPT in psychiatry. *Psychiatry Clin. Neurosci.* **2023**, *77*, 592–596.
22. Rice, S.; Crouse, S.R.; Winter, S.R.; Rice, C. The advantages and limitations of using ChatGPT to enhance technological research. *Technol. Soc.* **2024**, *76*, 102426.
23. Sakib, F.A.; Khan, S.H.; Karim, A.R. Extending the frontier of chatgpt: Code generation and debugging. In Proceedings of the 2024 International Conference on Electrical, Computer and Energy Technologies (ICECET), Sydney, Australia, 25–27 July 2024; IEEE: Piscataway, NJ, USA, 2024.
24. Dong, Y.; Jiang, X.; Jin, Z.; Li, G. Self-collaboration code generation via chatgpt. *ACM Trans. Softw. Eng. Methodol.* **2024**, *33*, 1–38.
25. Liu, Z.; Tang, Y.; Luo, X.; Zhou, Y.; Zhang, L.F. No need to lift a finger anymore? assessing the quality of code generation by chatgpt. *IEEE Trans. Softw. Eng.* **2024**, *50*, 1548–1584.
26. Feng, Y.; Vanam, S.; Cherukupally, M.; Zheng, W.; Qiu, M.; Chen, H. Investigating code generation performance of ChatGPT with crowdsourcing social data. In Proceedings of the 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC), Torino, Italy, 26–30 June 2023; IEEE: Piscataway, NJ, USA, 2023.
27. Fonseca, P.C.; Scherer, G.W. An image analysis procedure to quantify the air void system of mortar and concrete. *Mater. Struct.* **2015**, *48*, 3087–3098.
28. Shpak, A.; Gong, F.; Jacobsen, S. Frost durability of high-volume fly ash concrete: Relation liquid transport-damage. *Cem. Concr. Res.* **2023**, *163*, 107017.
29. Shpak, A. Production and Documentation of Frost Durable High-Volume Fly Ash Concrete: Air Entrainment, Cracking and Scaling in Performance Testing. Ph.D. Thesis, Norwegian University of Science and Technology, Trondheim, Norway, 2020.

30. Shpak, A.; Jacobsen, S. Effect of AEA-SP Dosage Sequence on Air Entrainment in FA Concrete. *Nord. Concr. Res.* **2019**, *61*, 1–21.
31. Thomas, M. *Optimizing the Use of Fly Ash in Concrete*; Portland Cement Association: Skokie, IL, USA, 2007; Volume 5420.
32. Du, L.; Folliard, K.J. Mechanisms of air entrainment in concrete. *Cem. Concr. Res.* **2005**, *35*, 1463–1471.
33. *NS-EN 206:2013+NA:2014*; (NO) National Annex NA (informative). Use of NS-EN 206:2013 in Norway [Translated from Norwegian]. Standards Norway: Lysaker, Norway, 2014.
34. Riley, N.A. Projection sphericity. *J. Sediment. Res.* **1941**, *11*, 94–95.
35. Gallagher, C.; Kerr, E.; McFadden, S. Particle size distribution for additive manufacturing powder using stereological corrections. *Powder Technol.* **2023**, *429*, 118873.