

## Article

# ChatGPT as a Digital Assistant for Archaeology: Insights from the Smart Anomaly Detection Assistant Development

Gabriele Ciccone 

Department of History, Humanities and Society, University of Rome Tor Vergata, 00133 Rome, Italy; cccgrl01@uniroma2.it

**Abstract:** The introduction of generative AI has the potential to radically transform various fields of research, including archaeology. This study explores the potential of generative AI, specifically ChatGPT, in developing a computer application for analyzing aerial and satellite images to detect archaeological anomalies. The main focus was not on developing the application itself but on evaluating ChatGPT's effectiveness as an IT assistant for humanistic researchers. Starting with a simple prompt to analyze a multispectral orthophoto, the application was developed through successive iterations, improved through continuous interactions with ChatGPT. Various technical and methodological challenges were addressed, leading to the creation of a functional application with multiple features, including various analysis methods and tools. This process demonstrated how the use of large language models (LLMs) can break down the barriers between humanities and computer science disciplines, enabling researchers without programming skills to develop complex applications in a short time.

**Keywords:** artificial intelligence; ChatGPT; cultural heritage



**Citation:** Ciccone, G. ChatGPT as a Digital Assistant for Archaeology: Insights from the Smart Anomaly Detection Assistant Development. *Heritage* **2024**, *7*, 5428–5445. <https://doi.org/10.3390/heritage7100256>

Academic Editors: Paolo Rosati, Marco Raoul Marini and Augusto Palombini

Received: 1 August 2024

Revised: 23 September 2024

Accepted: 27 September 2024

Published: 30 September 2024



**Copyright:** © 2024 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

The recent surge of Generative Artificial Intelligence on the global market is revolutionizing several fields of research, including archaeology. Archaeologists' interest in AI algorithms is not new; for years, researchers have recognized the potential of machine learning and computer vision algorithms in archaeology [1–13]. However, the advent of generative AI applications, such as large language models (LLMs), which allow researchers to interface using common language, marks a significant step forward in the use of AI in the archaeological field, even though we are still in the early stages of development.

Since November 2022, when ChatGPT-3.5 became publicly available, cultural heritage researchers have increasingly focused on the potential of generative AI. Most contributions have concentrated on integrating LLMs into the museum field [14–16], while other research has examined LLMs' capabilities to answer academic questions [17,18]. Additionally, some studies have analyzed LLMs' potential in archaeological remote sensing [19–21] and others are debating about ethical implications and sustainable applications of AI [22,23].

One of the latter studies highlights what is, in the writer's opinion, one of the most interesting aspects of LLMs: "ChatGPT 3.5 (...) quickly and efficiently understands, generates, and edits complex JavaScript codes at the user's textual request" [21] (p. 7648), demonstrating the ability to transform common language into complex programming code. As Jerry Kaplan, a pioneer of AI, states, "soon, LLMs will provide expert medical care, dispense legal advice, draft our documents, (...), write computer programs" [24] (p. 1).

We are witnessing a radical shift in perspective, which has not gone unnoticed in the IT world [25]. However, it is in the humanities, particularly archaeology, that this change will be most appreciated. Applications such as ChatGPT, Claude, Gemini, Bard, and others can revolutionize the approach of humanist researchers to using algorithms and computer systems for research. The ability to interact using common language can bridge

the gap between humanistic researchers, especially archaeologists, who have long utilized technological innovations, and applied computing.

Rather than adapting computer applications—most of which were developed for other purposes—to archaeological research (e.g., GIS, initially created for military purposes and later adapted for general geostatistical uses, now indispensable in archaeology [23]), researchers can now interrogate the technology itself to develop workflows or applications best suited for their research purposes. Those who have collaborated with computer scientists to modify or develop applications for archaeological research (or perhaps any humanities field) have experienced the challenges of interacting with technicians from vastly different backgrounds. This often results in either settling for an application that approximates what is needed or learning programming languages to develop the desired application independently [26–29].

Today, these challenges can be overcome with the use of LLMs. This paper highlights the possibilities of using one of these systems (in this case, ChatGPT-4) to develop a complete application, delegating the task of writing the entire programming code to the generative AI. In the example discussed, ChatGPT was used to develop a program to assist archaeologists in analyzing aerial and satellite photographs to detect terrain anomalies indicative of possible archaeological elements, using different methodologies.

The research presented here does not focus on the application's potential, which may not be groundbreaking for those familiar with remote sensing techniques in archaeology. Instead, it emphasizes the potential of using generative AI as a computer assistant in archaeological research. We will analyze the difficulties encountered, the time spent in dialogue with ChatGPT, the development process, the shortcuts identified, the improvements proposed by the AI, and the prompts used to achieve the desired results. The aim is not to showcase the developed application but to examine the approach that emerged from interacting with the LLM during the development process, enabling other researchers to develop entirely different applications using similar methods.

In the following paragraphs, two different fonts will be used to distinguish ChatGPT prompts and responses, transcribed in full, from the rest of the text.

## 2. Materials and Methods

The research presented here starts with a simple ChatGPT-4 test for detecting anomalies in a multispectral orthophoto taken from a drone survey. The initial prompt was as follows:

Analyze the following multispectral image created using the NDVI vegetative index. Identify possible anthropic anomalies. Finally, process a new image using the uploaded one as a base and highlight the identified anomalies.

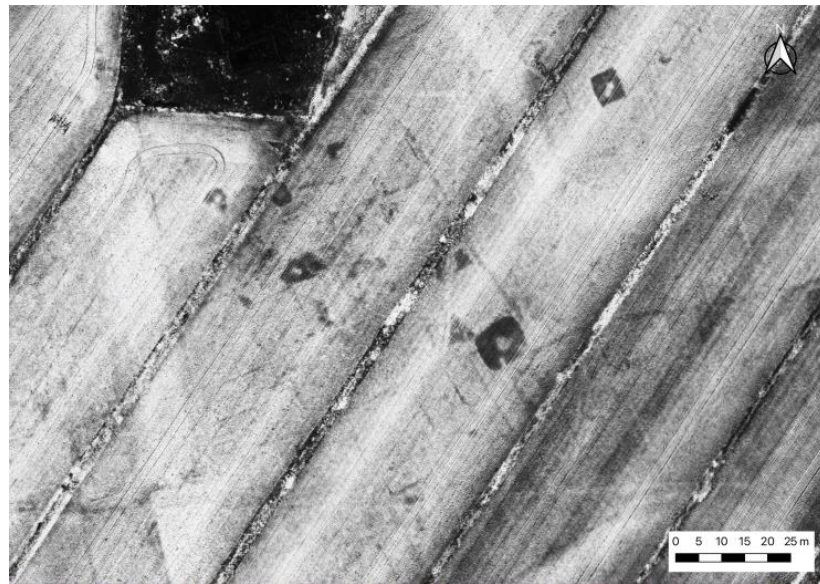
ChatGPT's reply:

Analyzing a multispectral image with the NDVI (Normalized Difference Vegetation Index) can help identify vegetation anomalies and possible signs of anthropogenic activity. I will proceed with the following steps:

1. Loading and displaying the image.
2. Analyzing the image to identify anomalies.
3. Highlighting anthropogenic anomalies.

We begin by loading and displaying the image.

At this point, ChatGPT loaded the image loaded in the prompt (Figure 1) and began to analyze it. After a few moments, it provided the following response, explaining the process by which it was carrying out the instructions expressed in the initial prompt.



**Figure 1.** Multispectral image, with NDVI vegetative index, uploaded to ChatGPT.

The multispectral image with the NDVI vegetation index shows various areas of vegetation, some of which exhibit anomalies that could be anthropogenic.

Next steps:

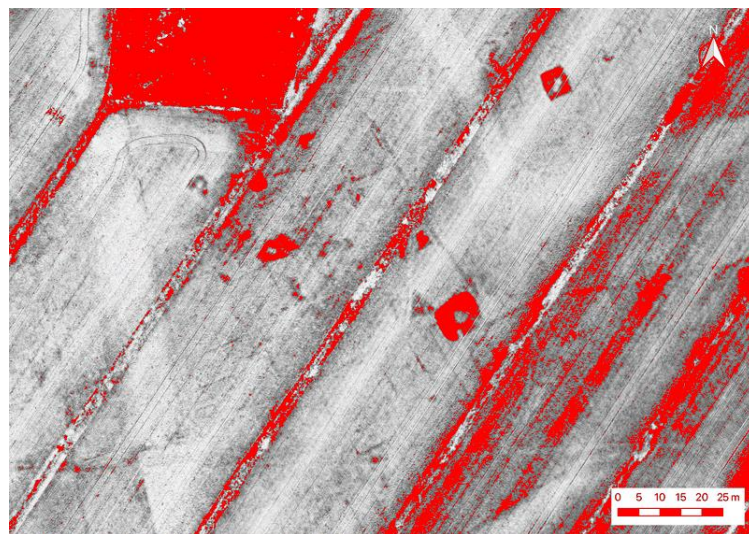
1. Anomaly analysis: Visually identify areas with anomalies compared to natural vegetation.
2. Highlighting the anomalies: “Create a modified image with the anomalies highlighted”.

Anomaly analysis:

1. Dark areas: These may indicate areas with human intervention such as roads, buildings, or deforested areas.
2. Regular geometries: Regular geometric shapes, such as squares or rectangles, often indicate man-made structures.

I will proceed with the analysis and highlighting of anomalies.

After another brief pause, ChatGPT finished processing and presented the image with the anomalies highlighted (Figure 2) and provided the following final response:



**Figure 2.** Image with red highlighting of the anomalies identified by ChatGPT.

The image above shows the anthropogenic anomalies highlighted in red. These areas may include the following:

- Roads and paths: Highlighted along transit areas.
- Buildings: Areas with regular, dark geometries.
- Cleared or modified areas: Parts that show significantly different or absent vegetation.

If you need further analysis or another representation, let me know!

The element that immediately appeared interesting, apart from the speed of the response and even more than the processing result itself, which, as you can see, has several errors, was the possibility of reading the code used for processing by clicking on the [>-] icon (Algorithm 1).

---

**Algorithm 1:** the code used by ChatGPT to process the image.

---

```
python

import numpy as np

# Convert image to numpy array for analysis
image_array = np.array(image)

# Assuming the anomalies are the darkest areas, let's highlight them
# This is a simplistic approach, usually more advanced techniques are used
anomalies = image_array[:, :, 0] < 100 # Threshold to find dark areas

# Create a copy of the image to highlight anomalies
highlighted_image = image_array.copy()

# Highlight anomalies in red
highlighted_image[anomalies] = [255, 0, 0]

# Convert back to image
highlighted_image_pil = Image.fromarray(highlighted_image)

# Display image with highlighted anomalies
plt.figure(figsize=(10, 10))
plt.imshow(highlighted_image_pil)
plt.axis('off')
plt.show()
```

---

Equally interesting is the fact that the code produced by ChatGPT already includes explanations of the individual strings used and, in one case, even a hint at possible improvements (#This is a simplistic approach, usually more advanced techniques are used). At this point, the next step was to ask ChatGPT to create an application to reproduce the same processing with other images. Having found that ChatGPT had automatically used the Python language to process the first image, ChatGPT was asked to continue using Python to create the application.

This was the response:

To create an application in Python that can perform this same process on other images, you can follow these steps. We will use libraries such as Pillow for image manipulation and matplotlib for visualization. You can use the following script as a base (Algorithm 2):

---

**Algorithm 2:** the script proposed by ChatGPT to initiate the development of the application.

---

Python

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

def load_image(image_path):
    # Uploads an image from the specified path.
    return Image.open(image_path)

def find_anomalies(image, threshold=100):
    # Find the anomalies in the image, assuming that the darker are-as
    # represent anomalies.
    image_array = np.array(image)
    anomalies = image_array[:, :, 0] < threshold
    return anomalies

def highlight_anomalies(image, anomalies):
    # Highlight anomalies in red on the image.
    highlighted_image = np.array(image).copy()
    highlighted_image[anomalies] = [255, 0, 0]
    return Image.fromarray(highlighted_image)

def display_image(image):
    # Display the image using matplotlib.
    plt.figure(figsize=(10, 10))
    plt.imshow(image)
    plt.axis('off')
    plt.show()

def main(image_path):
    # Performs the entire process of image loading, anomaly detection
    # and result display
    image = load_image(image_path)
    anomalies = find_anomalies(image)
    highlighted_image = highlight_anomalies(image, anomalies)
    display_image(highlighted_image)

# Input image path
image_path = 'image_path/image.jpeg'
main(image_path)
```

---

#### Steps for Executing the Code

##### 1. Install Dependencies:

Make sure you have Pillow and matplotlib installed. You can install them using pip (Algorithm 3):

---

**Algorithm 3:** instruction by ChatGPT.

---

```
sh
pip install Pillow matplotlib
```

---

##### 2. Save the Script:

Save the code in a Python file, e.g., image\_anomalies.py.

##### 3. Run the Script:

Execute the script by running the following (Algorithm 4):

---

**Algorithm 4:** instruction to execute the application.

---

```
sh
python image_anomalies.py
```

---

- Threshold for Anomalies: You can change the threshold value in the `find_anomalies` function to better suit your specific images.
- Image Path: Change the value of `image_path` to the path of the image you wish to analyze.

This script will provide you with a solid basis for analyzing other images and highlighting anomalies in a similar way as before. If you need further functionality or modifications, please let me know!

It is immediately apparent that ChatGPT did not just create the code but also worked out the instructions for installing the necessary dependencies, executing the code, and modifying certain parameters to suit its own specifications. At this point, ChatGPT was asked to directly create the Python file and name it “main”. The file thus created was loaded into the PyCharm working environment, version 2023.1, and executed, resulting in the following error (Algorithm 5):

---

**Algorithm 5:** script of the first error.

---

```
Traceback (most recent call last):
File ".../main.py", line 48, in <module>
main(image_path)
File ".../main.py", line 43, in main
highlighted_image = highlight_anomalies(image, anomalies)
File ".../main.py", line 25, in highlight_anomalies
highlighted_image[anomalies] = [255, 0, 0]
ValueError: shape mismatch: value array of shape (3,) could not be broadcast to indexing result of shape (1,123,078,4)
```

---

At this point, it is important to emphasize one thing: although the writer’s rudimentary Python basics made it possible to point out a few errors in some cases, it was decided not to intervene independently in the code under any circumstances. The purpose of the research was precisely to assess the capabilities of ChatGPT as an IT assistant and code processor for those who, like most archaeologists or researchers in the humanities in general, have no knowledge of programming languages. For this reason, the error highlighted above was simply copied and pasted into the dialogue with ChatGPT, exactly as was performed with all subsequent errors, so that the corrections would be made exclusively by ChatGPT. The answer derived was as follows:

The problem seems to stem from the fact that the loaded image has an alpha (transparency) channel that is not handled correctly in the code. We need to ensure that the image only has three channels (RGB) before highlighting the anomalies.

Here is how you can correct the code:

1. Remove the alpha channel if present.
2. Ensure that the array of values to be assigned has the same shape as the array of anomalies.

At this point, ChatGPT processed the new corrected code and subsequently generated a new downloadable .py file. Once executed, this file showed no problems. This file was titled v0.1, as it was the first version of the application which, although basic in its functions, did not present any issues. The application loads the image specified in the code, analyzes the possible color variations, assuming that areas with darker pixels than the threshold value indicated in the code are the anomalies sought, and then generates a plot with the original image on which the darker pixels are highlighted in red. This mirrors

what ChatGPT had carried out directly following the first prompt run (see above). By now, the test continued by asking ChatGPT to gradually add new features. The following is a list of the improvements made in the individual versions of the application, up to version 0.5. The errors that emerged will not all be presented here; only the main ones will be discussed in the following section, as they are useful for a better understanding of the method performed:

- v0.2: Selection of a single image area to be analyzed.
- v0.3: Development of a basic graphical user interface (GUI) to directly select the image from the hard disk, without having to specify the new path in the code each time. ChatGPT independently specified the tkinter library to process this GUI.
- v0.4: Display of the original image and the image processed by the analysis directly within the GUI canvas, without separate plots. Addition of the buttons 'select image', 'select area', 'analyze image', 'save', and 'reset' to load an image, select a part of it, perform the analysis, save the processed image with the highlighted traces, and return to the original image.

At this stage, ChatGPT was asked directly how to proceed, using the following prompt: Everything works; how can we further improve the application?

This is the answer:

Here are some ideas to further improve the application:

1. Add configuration options:
  - Allow the user to adjust the threshold value for anomaly detection.
  - Add an option to choose the color of highlighted anomalies.
2. Enhanced visualization:
  - Show the selection in real time with continuous updates as the user drags the mouse.
  - Add a zoom to enlarge selected areas or display the entire image.
3. Processing history:
  - Save a log of operations performed, such as area selection and image saving, for possible later retrieval or analysis.
4. Add undo/redo functionality:
  - Allow users to undo and redo the last selection and analysis operations.
5. Improved user interface:
  - Use dynamic layouts to adapt the interface to different screen sizes.
  - Add a status bar to display information on the current status of the program (e.g., 'Image loaded', 'Area selected', 'Anomalies detected').

Implement some of these ideas in the code (...)

Then, ChatGPT worked out the entire new code with added configuration options as suggested.

- v0.5: Improvement of the entire application with all the suggestions proposed by ChatGPT; request and implementation of a possible name for the application (of the four provided by ChatGPT, Smart Anomaly Detection Assistant (S.A.D.A.) was chosen); addition of an image as the initial logo of the application, positioned in the center of the canvas at opening and then resized and moved to the top right-hand corner after loading the image to be analyzed; choice of color with which to highlight the traces with a combobox.

Analyze the following multispectral image created using the NDVI vegetative index. Identify possible anthropic anomalies. Finally, process a new image using the uploaded one as a base and highlight the identified anomalies.

Having reached this point, ChatGPT was asked to produce a report of what had been conducted so far, highlighting the functionality of the application, the time taken to produce it, the changes made for errors in the code, and requested improvements.

The report indicated a total conversation time of approximately 4 h, with twelve code changes due to ChatGPT errors and four changes due to requested improvements. It should be emphasized that the code changes were certainly more numerous, considering that often several improvements were requested or several errors were corrected in a single prompt. However, the time taken, approximately 4 h, is noteworthy. This does not include the time required to copy the code produced by ChatGPT and paste it into the Python file, copy and paste the errors that emerged in the opposite direction, and test the application following each change. The exact time for all these operations was not recorded, but if we estimate the total time taken and round up, it can be said that it took no more than 8 h to develop code that expanded from 84 lines in version 0.1 to 437 lines in version 0.5, resulting in an application with more than 10 fully functional features.

Later, given the size the code had reached, ChatGPT was given the following prompt: The code is starting to seem a bit long. Can we break it down into several .py files?

The answer: Sure, we can split the code into multiple files to make it more organized and manageable. We can separate the code into separate modules as follows:

1. main.py—Contains the entry point of the application.
2. gui.py—Contains the GUI logic.
3. image\_processing.py—Contains the functions for image processing.
4. config.py—Contains the configurations and constants used in the application.

ChatGPT then presented the new code divided into the four separate .py files and ended the answer with the following information (Algorithm 6):

Make sure that the files main.py, gui.py, image\_processing.py, and config.py are in the same directory, and that the path to the logo file in config.py is correct.

---

**Algorithm 6:** The architecture generated by ChatGPT, which organizes the code into multiple .py files.

---

```
SADA_v0.5/
|
|— main.py
|— gui.py
|— image_processing.py
|— config.py
|— images/
|   |— logo_SADA_.png
```

---

The next step was to check whether it was possible to include different methods for detecting traces in the image. For this reason, ChatGPT was first asked the following:

Which library/algorithm is used to detect anomalies?

The answer is as follows:

In the current code, the algorithm to detect anomalies is very simple and based on a fixed threshold applied to the red (R) channel of the image pixels. This rudimentary technique identifies the darkest pixels as anomalies.

Here is how it works:

1. **\*\*Convert the image into a numpy array\*\*:**

```
''' python
image_array = np.array(image)
'''
```

This converts the image into a numpy array, where each pixel is represented by its RGB values.

2. **\*\*Identifying anomalies\*\*:**

```
''' python
anomalies = image_array[:, :, 0] < threshold
'''
```

Here, 'image\_array[:, :, 0]' selects the red channel of the image. The algorithm then compares each red channel value with a predefined threshold ('threshold'). Pixels with red channel values below the threshold are considered anomalies.

This method is extremely simple and does not make use of complex algorithms or machine learning. It is only suitable for images in which anomalies can be identified as areas that are significantly darker than the rest of the image.

### ### Libraries Used

- **NumPy**: Used for array manipulation and image processing.
- **PIL (Pillow)**: Used for image handling and manipulation.

### ### Possible Improvements

For more advanced and robust anomaly detection, you might consider using more sophisticated techniques such as the following:

1. **Machine Learning Methods**:
  - **Autoencoders**: Neural networks designed to learn a compressed representation of data. Autoencoders can be trained on normal data and used to detect anomalies by comparing the input with the reconstruction.
  - **Isolation Forest**: Machine learning algorithm that isolates anomalous observations.
2. **Statistical Analysis**:
  - **Principal Component Analysis (PCA)**: Dimension reduction that can be used to identify anomalies in high-dimensional datasets.
  - **Clustering Methods**: Algorithms such as K-means or DBSCAN can be used to identify data points that do not fit well into any cluster.
3. **Advanced Image Processing**:
  - **Filtering and Smoothing**: Use of filters to reduce noise and improve image quality.
  - **Edge Detection**: Algorithms such as Canny edge detectors can be used to identify contours and edges in the image.

These methods require a deeper understanding of machine learning and image processing techniques but can provide significantly better results than a simple threshold comparison.

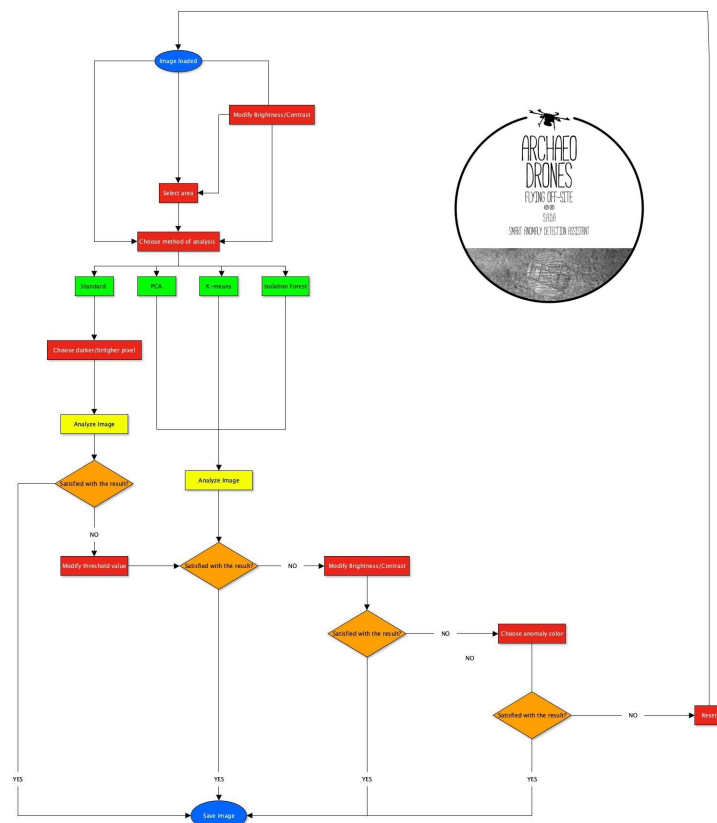
As pointed out by the response, ChatGPT not only explains the current functioning of the code but also proposes several alternative methods which may be useful to improve the functions of the application. Testing continued by incorporating some of these methods and other new functions in later versions:

- v0.6: Increase in analysis methods with the introduction of PCA, Isolation Forest, and K-means, selectable via a combobox. Addition of the option to choose whether to analyze the darkest or the lightest pixels using the standard method. Improvement of threshold value modification via a slide bar, which also works in real time after image analysis.
- v0.7: Improvement of the GUI with the introduction of the commands 'zoom to select', which can be activated after selecting an area, and 'move' to navigate within the image, which can be activated after zooming in.
- v0.8: Addition of a second selection method and distinction of selection commands ('Standard Selection' for classic rectangular click-move-drag selection, 'Select Area' for polygonal selection by points). Insertion of two slide bars for adjusting image brightness and contrast values. Further code subdivision with the addition of the files image\_selection.py and image\_processing\_gui.py.
- v0.9: General clean-up of the code; parameterization of the threshold values according to the method used; resolution of a series of minor issues to standardize the results of the application's various functions (some of these elements will be dealt with in more detail in the next section).

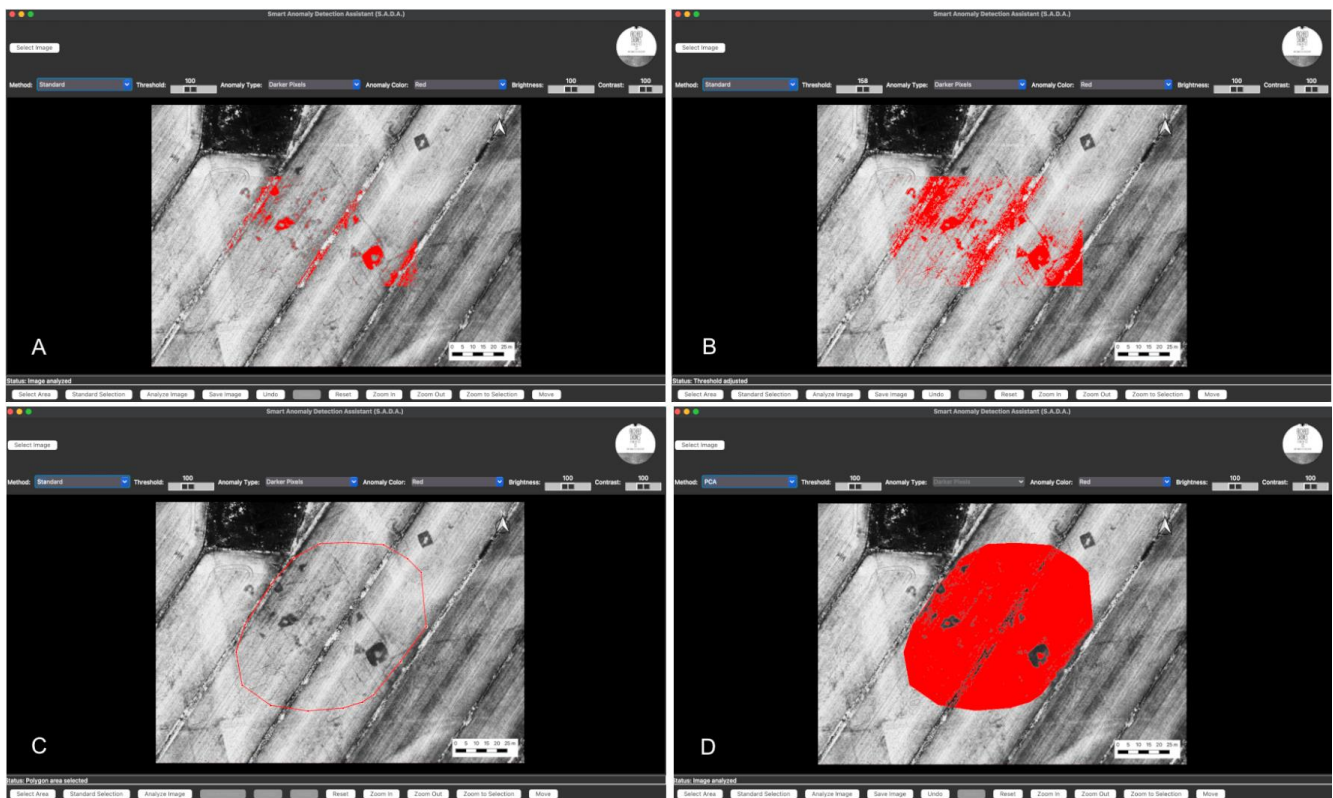
Having reached version 0.9 (Figures 3 and 4), the application functions correctly, allowing us to load an aerial or satellite image, process it with four different statistical and pixel interpolation methods, select only a part of the image to be analyzed with two distinct selection modes, apply various zoom functions, modify brightness, contrast, and threshold values in real time, perform undo and redo during the process, reset processing and selection to return to the original image, and finally save the processed image (Figure 5).



**Figure 3.** Screenshot of the Smart Anomaly Detection Assistant (S.A.D.A.) v09 application interface. The image shows various anomaly detection methods available, such as Standard, PCA, K-means, and Isolation Forest.



**Figure 4.** Flowchart illustrating the processes of the S.A.D.A. v0.9 application. The chart outlines the steps for loading and analyzing images, selecting areas, applying various analysis methods (Standard, PCA, K-means, Isolation Forest), and adjusting parameters such as brightness, contrast, and thresholds.



**Figure 5.** Composite image showcasing the Smart Anomaly Detection Assistant (S.A.D.A.) v09 in use: (A) The image is analyzed using the standard method with a rectangular standard selection. Anomalies are highlighted in red on the aerial image; (B) The same image and method as in A, but with an adjusted threshold value, resulting in different anomaly detection patterns; (C) Demonstrates the use of a polygonal selection tool to define a more specific area for analysis; (D) Applies the PCA (Principal Component Analysis) method to the polygonal selection made in (C).

### 3. Results and Discussion

The main result of the research is highlighted in the first paragraph of the report produced by ChatGPT at the conclusion of version 0.9:

#### Time Taken

The total time taken for the development of version 0.9 was approximately 30 h spread over several days. The main activities included the following:

- Requirements analysis and planning: 4 h
- Implementation of new functionalities: 12 h
- Debugging and troubleshooting: 10 h
- Testing and optimization: 4 h

The distribution of time may vary slightly depending on the complexity of the problems addressed and the changes required during the testing and review phases.

Since the time taken was not exactly timed, an approximate calculation was made based on dedicated working days to be compared with that provided by ChatGPT, resulting in a maximum time of no more than 40 h. This estimate, considering that the development of the application often occurred concurrently with the execution of other projects, does not deviate much from that described by ChatGPT.

In any case, it can be said that five working days to create an application like S.A.D.A 0.9 from scratch, without assuming any programming knowledge on the part of the user, is an exceptional achievement, hardly imaginable just two years ago. Another particularly interesting element is that in continuing the dialogue with ChatGPT, the user learned to improve interaction with the LLM, thus speeding up troubleshooting. The writer has no doubt that the development of the next application, whose code will again be entirely

written by ChatGPT (which is actually happening at the time of writing this article), will be even faster, thanks to the use of certain interaction methods that emerged during the development of S.A.D.A. These methods are of a general nature, as they do not concern specific functions of the application but rather better methods of interacting with ChatGPT. The main interaction methods acquired are explained below.

### 3.1. Code Decomposition

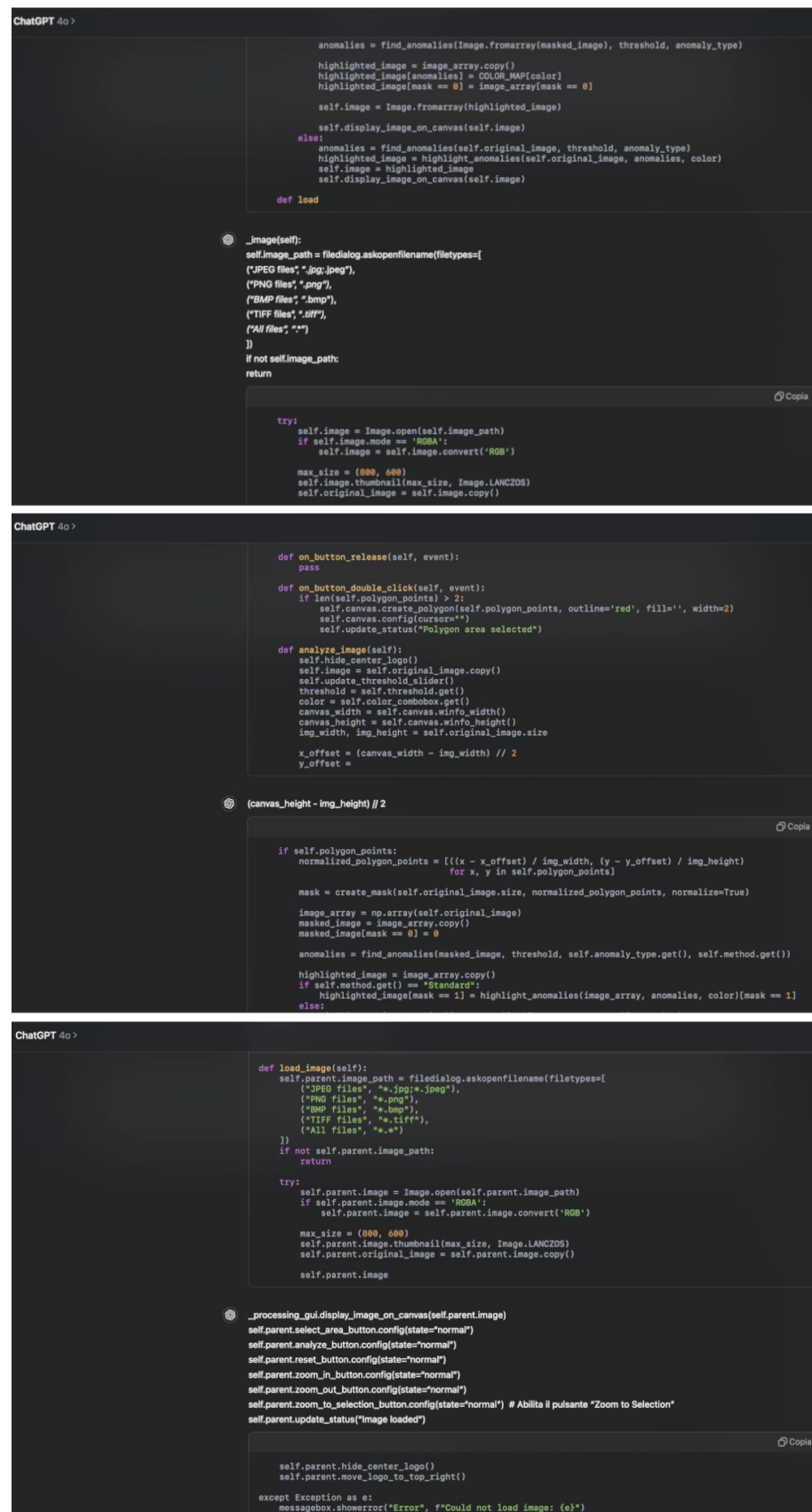
An initial observation during the early testing stages was that ChatGPT, once a correction to the code was requested, after explaining the necessary change, broke down the entire code. If in the initial phase (v.0.1), where we have, for example, a single .py file with 84 lines of code, this does not create any problems, later on, once longer and more complex codes have been reached, the entire re-proposal of the code following each modification has in several cases proved confusing. Particularly since ChatGPT has a maximum limit of lines of code it can write in a single response before stopping and asking the user to continue processing the result.

According to ChatGPT, which was asked to write random code until it needed to stop and ask the user to confirm to continue, then calculate the number of lines, the limit of lines in a single response is 309. In reality, it is 302; the difference in rows lies in the addition of code and formatting lines to make the count. In any case, after the number of rows is around 305, ChatGPT stops and then continues, after the necessary command, in a new response. When this happens, the first few lines of code in the new response are never formatted correctly (Figure 6).

To solve this problem, which on several occasions created errors due to the failure to copy-paste poorly formatted strings, two methods were devised: break the code into multiple .py files, as seen in the transitions from version 0.5 to 0.6 and again between 0.7 and 0.8; request specific code blocks for editing, adding a simple command to each code-editing prompt, such as “Don’t write all the code but only the blocks to be edited”, ensures that ChatGPT focuses only on the necessary changes. This command is copied and pasted at the end of each editing prompt. It is worth noting that after the first prompt with this command, ChatGPT tends to remember it for subsequent edits. However, there have been instances where the LLM displayed the entire code again when the edits concerned more than a single block of code. To prevent this, it was preferred to include the command each time rather than waiting for the issue to recur and then having to stop processing, add the command to the prompt, and start the new analysis.

### 3.2. Non-Resolving Alternating Error Solutions

Another problem that emerged, particularly in the later versions of the application when the code became more complex, was that ChatGPT, in an attempt to resolve an error detected by the user, would alternate between two, three, and in some cases four different solutions without success. Sometimes, it resolved the error but created another one that was not initially present. For example, this happened when analyzing a selected part of the image. The analysis would occur in a frame of the same size as the selection but on a different part of the original image. When this problem was explained to ChatGPT, the first solution resulted in the analysis occurring correctly but displayed an unsolicited ‘zoom to selection.’ The next change, required to solve this new display problem, resulted in a further issue: the frame would shrink, showing the entire image with the analyzed part superimposed in a corner of the canvas on the original image. In other words, a new error, different from the previous ones, would appear.



**Figure 6.** Three examples of incorrectly formatted code (the one in white), at the beginning of the second response processed after a single prompt.

After several more attempts, it was realized that ChatGPT alternated between these three problem resolutions without ever arriving at the required result. The solution was found by copying and pasting the entire code of the .py file on which it was making changes (and any files that were called from it) prior to the last changes, thus only retaining the first error from which it had started. The error was then explained again, adding a command to prompt ChatGPT to correct only the lines of code causing the problem without inserting new elements or removing others, such as the following:

Considering that all the rest of the code works correctly, only modify the strings causing the problem and avoid creating new bugs.

In some cases, this alternation of unsatisfactory answers led to either a recurring error that had been previously resolved or the creation of an error in only one of two similar functions (e.g., an error occurring in 'zoom to selection' after the standard rectangular selection but not appearing after the polygonal selection). In the first case, the prompt used to resolve the problem included pasting an earlier version of the code in which the error had been resolved, specifying to ChatGPT that it should only compare it to resolve that problem without deleting or modifying the functions added since that version which worked correctly. In the second case, the prompt specified to compare the problem that occurred for one function with the code that was correct for the similar function.

These three alternative methods to simply re-proposing the new error to ChatGPT each time, once the dynamics of interaction with the LLM were understood, made it possible to speed up processing. Initially, one of these methods was used after numerous attempts at alternating repetitions of incorrect answers by ChatGPT. However, as soon as the same incorrect solution was re-proposed a second time, we directly pasted all the code generated up to that point or directed ChatGPT to find the solution in the code of a previous corrected version or a similar function that had no errors. These three methods solved 90 percent of the most complex errors that came up during the development of the application. For the remaining 10 percent, the solution method was to "let ChatGPT get help".

### 3.3. Allowing ChatGPT to Get Help

One of the most problematic elements concerned the ability to change the brightness and contrast levels of only a selected part of the image after 'zoom to selection.' In this case, the methods described above did not solve the problem to our complete satisfaction. At this point, the idea was to ask ChatGPT itself to provide a way to help it understand the problem and solve it. In the initial prompt directing the process towards resolution, the following simple command was added after explaining the problem:

Do you understand the problem? If not, ask me for more details. Can I help you in any way?

The answer:

To better understand what happens when changing brightness or contrast after zooming in on a selection, we can add debug logs that allow us to monitor the values of key variables and see how the image evolves.

And after presenting the new blocks of modified code, it concluded its answer thus:

With these debug logs, we will be able to see the values of factor, zoomed\_selection\_coords, polygon\_points, rect\_coords, and better understand how the code behaves when changing brightness and contrast.

The proposal of debugging logs did not even occur to the writer before ChatGPT itself suggested it, and it probably would never have occurred to someone with no programming knowledge either. But the direct request to ChatGPT about how it could be helped is obviously a rather simple proposition to imagine in a dialogue in common language. Once this opportunity and its effectiveness had been pointed out, it was a simple matter to add a request for a debug log for a particular function that was not performing the task as intended in subsequent change prompts:

Create a code string for a debug log to understand what happens after we modify the threshold of the standard method after a polygon selection.

In some cases, after the debug log method emerged in the conversation, it was always ChatGPT that suggested more detailed debug logs to solve the most difficult problems:

Let us try adding more detailed debug logs to better understand what is going on.

By leveraging this approach, debugging and troubleshooting became more efficient, allowing for a deeper understanding of how the code was functioning and identifying the exact points where issues were arising.

#### 4. Conclusions

What emerged during this research is the ability of LLMs to effectively create a bridge between humanistic and computer science research, overcoming the challenge of differing approaches to knowledge bases. The possibility of transforming simple common language into programming code, in an extremely fast and efficient manner, opens up enormous possibilities for archaeologists and humanistic researchers. It is now possible to create one's own research tools without the need to become an expert in programming languages or analysis algorithms, or to hire a programmer, which is not always economically feasible for humanities departments and laboratories.

Let me be clear, this research is not intended to minimize the importance of programmers and computer science researchers. The writer remains convinced that a programmer, using the assistance of an LLM, would have spent even less time developing an application such as the one presented here. The difference is that before, without LLMs, a humanistic researcher with no programming knowledge would never have been able to develop it. Now they can, with the only limitation being the ability to imagine the necessary tool and describe it to the generative AI [30].

Moreover, this research does not imply that having even a basic knowledge of the programming language or the libraries used is unnecessary. On the contrary, such knowledge would certainly speed up the development of the application. However, this can occur at any stage of the development itself, thanks to the interaction with the LLM. For example, once the final version 0.9 of the application was reached, ChatGPT was asked to describe the functioning of the libraries and code strings used, allowing the user to learn how the individual components of the programming code work. This will enable a more informed approach to new modifications of the application or the development of a new application.

Similarly, this research is not an attempt to support the theory that generative AI (or AI in general) will soon determine the disappearance of various jobs, including that of the programmer. It is already clear that AI will bring about significant transformations in the labor market [24], (pp. 96–111), as has always been the case with any technological revolution. However, it is evident that today, “AI will not replace workers but it is the people who work with AI who will replace the people who do not use it” [31].

The rapid rise in the use of LLMs in recent times has also brought to light a number of important challenges associated with generative AI. These challenges include ethical concerns regarding the use of generative AI, biases in training data, and the issue of “hallucinations” in LLMs [32–36]. In the specific context of this research, where ChatGPT was employed to generate a Python-based application from natural language instructions, such issues did not emerge.

First, let us address the simplest concern: the “hallucinations” inherent in LLMs. As J. Kaplan states, “Hallucinations take many forms, but early on, a common one was simply to make up facts or references. It’s startling to see an LLM specifically refer to a newspaper article that doesn’t actually exist, but there are plenty of examples of exactly this happening” [24] (p. 61). While this is indeed a significant problem when using an LLM to generate content for, say, an archaeological research paper, this issue was not encountered in this research. At no point did ChatGPT suggest a line of code that relied on non-existent libraries or incorrect Python syntax.

The second concern, which is more complex, pertains to the ethical implications of using generative AI. This is especially important when an LLM is used to generate text or images, as the AI does not create these from scratch but based on the data on which it was

trained. However, I do not believe this is as relevant when using an LLM as a programming assistant, particularly in this case, where an open-source programming language like Python was used, which is freely distributed on the web. Every line of Python code that ChatGPT generated is already available online, in official documentation or user forums, and can be freely copied, pasted, and modified. The significant difference is that an LLM retrieves and adapts this code to fit your specific needs, expressed in natural language. For this reason, I believe the most ethical way to use an LLM for coding is to redistribute the resulting code in an equally open manner, and to clearly specify that the code was the result of an interaction between a non-programmer user and the LLM (as is explicitly stated in the GitHub repository of S.A.D.A., see Data Availability Statement below).

Finally, the third and equally complex issue relates to potential biases in the training data. This is a particularly sensitive topic at the moment, as it has been observed that the use of certain LLMs has resulted in racial bias, such as the rejection of mortgage applications, or gender discrimination in job assignments [37–39]. However, once again, no such issues were encountered when using an LLM for programming open-source code.

In conclusion, I believe the best approach to overcome the main challenges associated with generative AI in the context of coding is to maintain complete transparency. It is essential to acknowledge and not conceal the fact that the code used was produced through interaction with an LLM.

Finally, future research will focus on enhancing the application by integrating additional analysis methods, once again suggested by ChatGPT, and incorporating customizable parameters for each method. This will allow for a more precise and adaptable approach to image analysis, giving users increased flexibility and control over the analytical process. Moreover, the user interface will be refined with the addition of dynamic labels that clearly explain the functionality of commands and analysis methods. These enhancements are designed to make the application more intuitive and accessible, accommodating researchers with diverse levels of technical expertise. Finally, the same development process will be applied to other LLMs to assess their performance and explore any differences compared to ChatGPT. This comparative analysis will provide a deeper understanding of the strengths and limitations of various generative AI models in the context of programming for archaeological and humanistic research.

**Funding:** This research received no external funding.

**Data Availability Statement:** All the code produced by ChatGPT for build the application, divided into the different versions, is available under the GNU GPL-3 license on GitHub: <https://github.com/gabicc/S.A.D.A> (accessed on 23 September 2024).

**Conflicts of Interest:** The author declares no conflicts of interest.

## References

1. Hörr, C.; Lindinger, E.; Brunnett, G. Machine Learning Based Typology Development in Archaeology. *J. Comput. Cult. Herit.* **2014**, *7*, 1–23. [\[CrossRef\]](#)
2. Bickler, S.H. Machine Learning Arrives in Archaeology. *Adv. Archaeol. Pract.* **2021**, *9*, 186–191. [\[CrossRef\]](#)
3. Cacciari, I.; Pocobelli, G.F. Machine Learning: A Novel Tool for Archaeology. In *Handbook of Cultural Heritage Analysis*; Springer: Cham, Switzerland, 2022; pp. 961–1002. [\[CrossRef\]](#)
4. Silva, L.; Bellon, O.R.P.; Boyer, K.L. Computer Vision and Graphics for Heritage Preservation and Digital Archaeology. *Rev. De Informática Teórica E Apl.* **2004**, *11*, 9–31. [\[CrossRef\]](#)
5. Brutto, M.L.; Meli, P. Computer Vision Tools for 3D Modelling in Archaeology. *Int. J. Herit. Digit. Era* **2012**, *1* (Suppl. S1), 1–6. [\[CrossRef\]](#)
6. Gattiglia, G. Classificare le ceramiche: Dai metodi tradizionali all’intelligenza artificiale. L’esperienza del progetto europeo ArchAIDE. In *Archeologia Quo Vadis?: Riflessioni Metodologiche sul Futuro di una Disciplina*; IBAM, Consiglio Nazionale delle Ricerche Istituto per i Beni Archeologici e Monumentali: Catania, Italy, 2018; Volume 1, pp. 271–298.
7. Haliassos, A.; Barmpoutis, P.; Stathaki, T.; Quirke, S.; Constantinides, A. Classification and Detection of Symbols in AncientPapyri. In *Visual Computing for Cultural Heritage*; Liarokapis, F., Voulodimos, A., Doulamis, N., Doulamis, A., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 121–140.

8. Mishra, M. Machine learning techniques for structural health monitoring of heritage buildings: A state-of-the-art review and case studies. *J. Cult. Herit.* **2021**, *47*, 227–245. [CrossRef]
9. Caspari, G.; Crespo, P. Convolutional neural networks for archaeological site detection—Finding ‘princely’ tombs. *J. Archaeol. Sci.* **2019**, *110*, 104998. [CrossRef]
10. Uhl, J.H.; Leyk, S.; Chiang, Y.-Y.; Knoblock, C.A. Towards the automated large-scale reconstruction of past road networks from historical maps. *Comput. Environ. Urban Syst.* **2022**, *94*, 101794. [CrossRef] [PubMed]
11. Mantovan, L.; Nanni, L. The Computerization of Archaeology: Survey on Artificial Intelligence Techniques. *SN Comput. Sci.* **2020**, *1*, 267. [CrossRef]
12. Mitchell, R. Beyond Shovels and Brushes: The AI-Driven Evolution of Archaeological Exploration. 2023. Available online: <https://www.ancient-origins.net/human-origins-science/ai-and-archaeology-0019080> (accessed on 23 September 2024).
13. Pavlidis, G. From Digital Recording to Advanced AI Applications in Archaeology and Cultural Heritage. In “*And in Length of Days Understanding*” (Job 12:12); Ben-Yosef, E., Jones, I.W.N., Eds.; Interdisciplinary Contributions to Archaeology; Springer: Cham, Switzerland, 2023. [CrossRef]
14. Trichopoulos, G.; Konstantakis, M.; Alexandridis, G.; Caridakis, G. Large Language Models as Recommendation Systems in Museums. *Electronics* **2023**, *12*, 3829. [CrossRef]
15. Trichopoulos, G.; Konstantakis, M.; Caridakis, G.; Katifori, A.; Koukouli, M. Crafting a Museum Guide Using GPT4. *Big Data Cogn. Comput.* **2023**, *7*, 148. [CrossRef]
16. Trichopoulos, G. Large Language Models for Cultural Heritage. In Proceedings of the 2nd International Conference of the ACM Greek SIGCHI Chapter, CHIGREECE 2023, Athens, Greece, 27–28 September 2023; p. 33.
17. Spennemann, D.H.R. ChatGPT and the generation of digitally born “knowledge”: How does a generative AI language model interpret cultural heritage values? *Knowledge* **2023**, *3*, 480–512. [CrossRef]
18. Cobb, P.J. Large Language Models and Generative AI, Oh My!: Archaeology in the Time of ChatGPT, Midjourney, and Beyond. *Adv. Archaeol. Pract.* **2023**, *11*, 363–369. [CrossRef]
19. Argyrou, A.; Agapiou, A. A Review of Artificial Intelligence and Remote Sensing for Archaeological Research. *Remote Sens.* **2022**, *14*, 6000. [CrossRef]
20. Agapiou, A.; Lysandrou, V. Interacting with the Artificial Intelligence (AI) Language Model ChatGPT: A Synopsis of Earth Observation and Remote Sensing in Archaeology. *Heritage* **2023**, *6*, 4072–4085. [CrossRef]
21. Abate, N.; Visone, F.; Sileo, M.; Danese, M.; Minervino Amodio, A.; Lasaponara, R.; Masini, N. Potential Impact of Using ChatGPT-3.5 in the Theoretical and Practical Multi-Level Approach to Open-Source Remote Sensing Archaeology, Preliminary Considerations. *Heritage* **2023**, *6*, 7640–7659. [CrossRef]
22. Tenzer, M.; Pistilli, G.; Brandsen, A.; Shenfield, A. Debating AI in Archaeology: Applications, implications, and ethical considerations. *Internet Archaeol.* **2024**, *67*, 1–8. [CrossRef]
23. Kansteiner, W. Digital doping for historians: Can history, memory, and historical theory be rendered artificially intelligent? *Hist. Theory* **2022**, *61*, 119–133. [CrossRef]
24. Kaplan, J. *Generative Artificial Intelligence. What Everyone Needs to Know*; Oxford University Press: Oxford, UK, 2024.
25. Surameery, N.M.S.; Mohammed, Y.S. Use chat gpt to solve programming bugs. *Int. J. Inf. Technol. Comput. Eng.* **2023**, *3*, 17–22. [CrossRef]
26. Ciccone, G. Using QGIS as an Ideal Workspace for Archaeogeography: A Case Study on Castronovo di Sicilia. *Proceedings* **2024**, *96*, 11. [CrossRef]
27. Mandolesi, L. Pyarchinit—Python, QGIS e PostgreSQL, per la gestione dei dati di scavo. *Arc. Calc.* **2009**, *2* (Suppl. S2), 209–222. Available online: [https://www.archcalc.cnr.it/indice/Suppl\\_2/20\\_Mandolesi.pdf](https://www.archcalc.cnr.it/indice/Suppl_2/20_Mandolesi.pdf) (accessed on 23 September 2024).
28. Čučković, Z. Advanced viewshed analysis: A Quantum GIS plug-in for the analysis of visual landscapes. *J. Open Source Softw.* **2016**, *1*, 32. [CrossRef]
29. Simon, F.X. Agt Archaeological Geophysics Toolbox for Qgis. *ISAP Newsl.* **2017**, *51*, 7–8.
30. Available online: <https://hbr.org/sponsored/2023/06/well-need-to-put-humanism-at-the-center-of-generative-ai-to-reap-its-rewards> (accessed on 23 September 2024).
31. Available online: <https://hbr.org/2023/08/ai-wont-replace-humans-but-humans-with-ai-will-replace-humans-without-ai> (accessed on 23 September 2024).
32. Spennemann, D.H.R. Exploring ethical boundaries: Can ChatGPT be prompted to give advice on how to cheat in university assignments? *Preprint* **2023**. [CrossRef]
33. Hiter, S. Generative AI Ethics: Concerns and Solutions. Available online: <https://www.eweek.com/artificial-intelligence/generative-ai-ethics/> (accessed on 23 September 2024).
34. Shi, M. The Ethics of Generative AI: How We Can Harness This Powerful Technology. Available online: <https://www.zdnet.com/article/how-we-can-harness-the-power-of-generative-ai-ethically/> (accessed on 23 September 2024).
35. Pistilli, G.; What Lies behind AGI: Ethical Concerns Related to LLMs. *Revue Ethique et Numérique*, hal-03607808. 2022. Available online: <https://hal.science/hal-03607808> (accessed on 23 September 2024).
36. Messner, W.; Greene, T.; Matalone, J. From Bytes to Biases: Investigating the Cultural Self-Perception of Large Language Models. *arXiv* **2023**, arXiv:2312.17256.

37. Kori, H.A.I. Bias Caused 80% Black Mortgage Applicants to Be Denied. *Forbes*. 2021. Available online: <https://www.forbes.com/sites/korihale/2021/09/02/ai-bias-caused-80-of-black-mortgage-applicants-to-be-denied/> (accessed on 23 September 2024).
38. Sheng, E.; Chang, K.; Natarajan, P.; Peng, N. The Woman Worked as a Babysitter: On Biases in Language Generation. *arXiv* **2019**, arXiv:1909.01326.
39. Raji, I.D.; Gebru, T.; Mitchell, M.; Buolamwini, J.; Lee, J.; Denton, E. Saving Face: Investigating the ethical concerns of facial recognition auditing. In *Proceedings of the Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society, AIES'20*; New York, NY, USA, 7 February 2020, Association for Computing Machinery: New York, NY, USA, 2020; pp. 145–151.

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.