CHATGPT FOR POINT CLOUD 3D OBJECT PROCESSING

J. Balado[a,*], G. Nguyen[b]

[a]GeoTECH, CINTECX, Universidade de Vigo, 36310 Vigo, Spain (jbalado@uvigo.gal)
[b]Faculty of Informatics and Information Technologies, Slovak University of Technology, Ilkovičova 2, Bratislava 84216, Slovakia (giang.nguyen@stuba.sk)

KEY WORDS: Language Models, Artificial Intelligence, LiDAR, Natural Language Processing, geometric features, eigenvalues.

ABSTRACT:

Large-scale pretrained language models have been a revolution in human-machine communication. Recently, such language models also generate code for required tasks. The objective of this work is to evaluate the functionality of the codes generated by ChatGPT (version 15-Dec-2022) for point cloud processing. The programming language selected for the test was MATLAB due to the extensive use in prototyping and toolboxes for Computer Vision and LiDAR. Using the Question-Answer system, the ChatGPT was asked for codes to calculate surface normals, curvature, eigenvalues, and eigenfeatures, with specific parameters and outputs. The provided codes were compiled and executed. The results show that ChatGPT can generate functional code for very specific and short applications, however, it is not capable of generating large code involving the correct use of loops, indexes, or equations.

1. INTRODUCTION

Point cloud processing is a crucial technology for 3D object detection and mapping. A point cloud is a set of points in 3D space that represent the surface of an object or scene. These points are usually acquired using 3D sensors such as lidar or stereo cameras, and they provide a detailed representation of the geometry and structure of the object or scene.

Point cloud processing refers to the algorithms and techniques used to analyse, manipulate, and extract information from point clouds. This includes tasks such as segmentation, registration, feature extraction, and classification. These techniques are essential for 3D object detection and mapping, as they allow us to identify and locate objects in a scene, and to create accurate and detailed 3D maps of the environment.

Point cloud processing has many applications in fields such as robotics, autonomous vehicles, augmented and virtual reality, and computer vision. It allows us to accurately perceive and understand the 3D world around us, and to interact with it in a more natural and intuitive way. It also has important implications for fields such as security, surveillance, and environmental monitoring, where the ability to detect and track objects in 3D is crucial.

Overall, point cloud processing is a key technology for 3D object detection and mapping, and it has numerous applications and implications in a variety of fields.

The above introduction was ChatGPT's response to the question *Can you write an introduction on the relevance of point cloud processing for 3D object detection and mapping?* Although a bit repetitive emphasizing the importance of point clouds, the introduction was automatically structured on a correct definition of point clouds, the main tasks that compose processing (Che et al., 2019; Wang and Kim, 2019; Xia et al., 2020), and their applications (Li et al., 2021; Liu et al., 2019; Xie et al., 2020).

ChatGPT is, in its own words, *a large artificial neural network that has been trained on a vast amount of text data to generate human-like responses to questions and prompts. Its primary function is to assist with a variety of tasks by providing information and answering questions to the best of its ability* (answer to the question *In technical words, who are you?*).

Unlike other chatbots, ChatGPT is based on the GPT-3 AI language model, developed by OpenAI, and is not only able to provide more human-like responses to a conversation, but can also generate text and code in the desired programming language.

The objective of this work is to evaluate the use of GPT chat to obtain the codes for point cloud processing in MATLAB, such as normal estimation or geometric feature extraction. This way it is evaluated if the proposed codes are functional and to what extent they can be useful to people with different degrees of knowledge in point cloud processing.

The rest of this paper is organized as follows. In Section 2, the state of the art on Artificial Intelligence chats is collected. In Section 3, ChatGPT and the methodology of questions-answers are explained. Section 4 presents, analyses and discuss the results of scripts prosed by the chat. Section 5 concludes the work.

2. STATE OF THE ART
LARGE-SCALE PRETRAINED LANGUAGE MODELS

The understanding and the treatment of the ubiquitous textual data is a major research challenge when the tremendous amount of such data produced by our society has exploded over the recent years. Natural Language Processing (NLP) aims to provide a set of techniques able to explain a wide variety of natural language tasks in the meaning extraction process to be successful. Recurrent Neural Networks (RNN) were massively used to solve NLP problems, especially with the success of Long Short-Term

* Corresponding author

Memory (LSTM) and Gated Recurrent Unit (GRU) architectures in preventing the vanishing gradient issue in training. However, those two models are also not perfect because of the recurrent structure made them hard to parallelize and the treatment of very long clauses is also problematic (Gillioz et al., 2020).

To counter those limiting constraints, the Transformer architecture was introduced with the massively use of unsupervised pre-training principle (Gillioz et al., 2020). The pre-training principle works in two steps: 1) the pre-training phase, which computes a general representation from raw data in an unsupervised fashion; 2) the use of finetuning techniques to adapt to a concrete task. The original Transformer used an encoder–decoder architecture like earlier sequence-to-sequence models with adoption of the self-attention mechanism, differentially weighting the significance of each part of the input data. The recent significant increase in the performance of NLP models is due to the use of word embeddings and the attention mechanism aims to catch the long-term dependencies of sentences. Mathematically, Transformer takes a sequence $X=(x_1, \ldots, x_N)$ and produce a latent representation $Z=(z_1, \ldots, z_N)$, then the output sequence $Y_M=(y_1, \ldots, y_M)$ is produced one element at a time. This property is called the autoregressive property of the model, where a model takes the previous outputs to produce the next outcome.

After that, various models coupled with the Transformer re using this property to produce accurate Language Model languages like ELMo (by Allen Institute for AI), BERT (Bidirectional Encoder Representations from Transformers), GPT (Generative Pre-trained Transformer), and GPT-2 (by OpenAI). Due to the high performance of BERT and the need of larger training datasets, there are a lot of post-BERT models (like RoBERTa, DistilBERT, AIBERT) aiming to tweaked it to specific needs such as simpler architecture, smaller models, reducing number of layers without a loss of performance.

In the context with GPT, it is suitable to mention that OpenAI was founded as a non-profit research organisation in 2015 with the mission is to ensure that artificial general intelligence (AGI) benefits all of humanity. In 2019, OpenAI had restructured as a profit company and in 2020, despite various own previous concerns about malicious applications, it announced GPT-3, a new language model more than 100 times larger than GPT-2, with 175B parameters and 96 layers trained on a corpus of 499B tokens of web content (Dale, 2021). The model itself was not made available, but the access was provided via an API giving the model's creators more control over its use (e.g., waiting list, high price). The typical use of the API is a prompt with an initial text to get the model going, along with optional parameter setting (Floridi and Chiriatti, 2020).

Several produced outputs are truly breath-taking as candidates for being human-authored text. The technology has also been lauded for its results in a wide range of other areas such as capability in generating poetry, playing chess, doing arithmetic, and writing programming code based on requirements expressed in natural language. The technology made incredible, amazing, and challenging impression. It is also threatening, which can be expressed shortly as "*even with 175B parameters and 450 gigabytes of input data, **it's not a reliable** interpreter of the world*" (Gary Marcus and Ernest Davis, MIT Technology Review). After a hype and the conclusion (not reliable narrator), this is not to say that GPT-3 is devoid of practical application; meaning some use cases are appropriate and some are not (Zhang and Li, 2021) and unreliable doesn't mean useless. As the GPT-3 model presents security and uncontrollability problems, including false content and biased information during content generation, its application value is primarily reflected in intelligent auxiliary tasks, and it cannot directly interface with the end-users.

3. METHODOLOGY

3.1 ChatGPT

The GPT-3 model offers many powerful functions that can cope with many practical application scenarios, such as question answering, reading comprehension, summary generation, automatic chat, search matching, code generation, and article generation (Zhang and Li, 2021). Recently, ChatGPT is fine-tuned from a model in the GPT-3.5 series (Yao Fu et al., 2022), which finished training in early 2022 on an Azure AI supercomputing infrastructure. It is a sibling model to (OpenAI) InstructGPT model, which is trained to follow an instruction in a prompt (Figure 1) and provide a detailed response under the Reinforcement Learning from Human Feedback (RLHF) approach (OpenAI, 2022).

Despite its powerful modelling and description capabilities, there are significant issues and limitations. Such GPT-a-like model does not understand writing well and sometimes generates uncontrollable content. Secondly, training such models requires a large amount of computing power, data, and capital investments. The most important thing is although ChatGPT is trained with human-in-loop, it is just more truthful and less toxic.

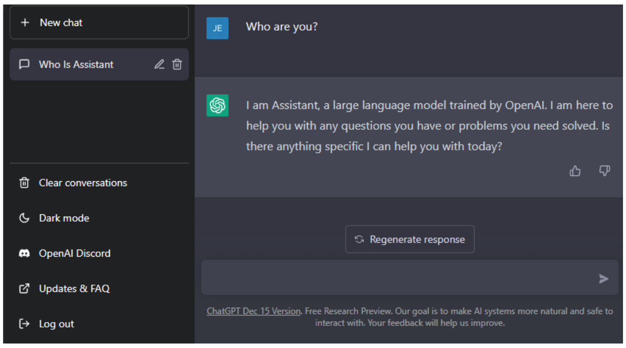

Figure 1. ChatGPT user interface.

3.2 Question and Answer process

The way to obtain information from ChatGPT is through a process of queries and responses (Figure 2). The first question is the generation of a code for the execution of the desired operation. Then, the answer is evaluated to see if it matches the desired method. If it is observed that the code has no relation with the desired one, or if no code is generated, the Chat is instructed to try again, either by making variations in the question to focus the result or not. If the code looks like the desired one, it is tested in MATLAB with a point cloud. If the code generates an error, it is indicated to the chat in question mode. When a code that generates the desired output is obtained, modifications in parameters or outputs are applied.

In order to evaluate code generation, some basic point cloud processing methods to extract feature information are selected to be generated automatically by ChatGPT. These features are the basis of point cloud processing for object segmentation and identification. Given a 3D point cloud $X = \{p_1, p_2, \cdots, p_N\} \in R^{N\times 3}$, with for each local sample $P_i = \{p_{i,j} \mid p_{i,j} \in KNN(p_i)\}$, constructed by the k nearest neighbour (kNN) search (Zhou et al., 2021). The covariance matrix of the local neighbourhood can be used to estimate local surface properties (Pauly et al., 2002). The covariance matrix for a point p is given in equation (1) where $\bar{p}$ is the centroid of the neighbourhood.

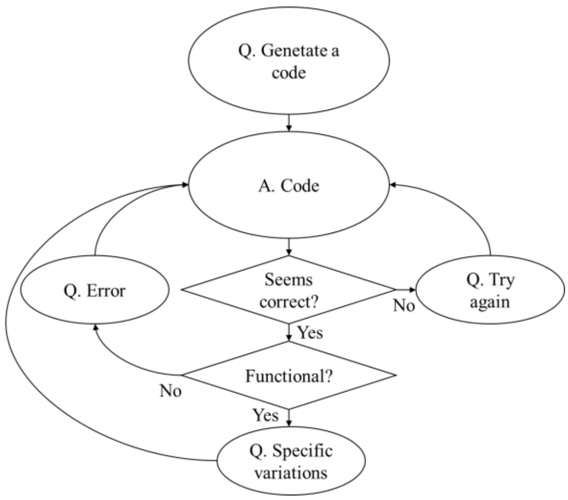

Figure 2. Workflow of questions (Q.) and answers (A.) to generate functional code.

$$C = \begin{bmatrix} p_{i_1} - \bar{p} \\ \cdots \\ p_{i_k} - \bar{p} \end{bmatrix}^T \cdot \begin{bmatrix} p_{i_1} - \bar{p} \\ \cdots \\ p_{i_k} - \bar{p} \end{bmatrix}, i_j \in N_p \tag{1}$$

Consider the eigenvector problem in equation (2) and since the covariance matrix C is symmetric and positive semi-definite, all eigenvalues λ are real-valued and the eigenvectors v form an orthogonal frame, corresponding to the principal components of the point set (Jolliffe, 2002). The λ measure the variation of the p_i along the direction of the corresponding eigenvectors.

$$C \cdot v_l = \lambda_l \cdot v_l, \qquad l \in \{1,2,3\} \tag{2}$$

Therefore, being $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$ with a plane $T(x): (x - \bar{p}) \cdot v_3 = 0$. The v_3 approximates the surface normal n_p at p_i and v_1 and v_2 are tangent vectors of plane T at p_i.

The curvature describes the surface variation along surface normal in equation (3). Alternatively, it can also be calculated as the variance of the normal, as shown in equation (4).

$$curvature\_\lambda = \frac{\lambda_3}{\lambda_l + \lambda_2 + \lambda_3} \tag{3}$$

$$curvature\_n = \frac{\sum_1^k \left(n_{i_j} - \bar{n}\right)^2}{k} \tag{4}$$

Other eigenfeatures describe de dimensionality of point spatially distribution (Weinmann et al., 2015), such as linearity in equation (5), planarity equation (6) and scattering equation (7).

$$\text{linearity} = \frac{\lambda_1 - \lambda_2}{\lambda_l} \tag{5}$$

$$\text{planarity} = \frac{\lambda_2 - \lambda_3}{\lambda_l} \tag{6}$$

$$\text{scattering} = \frac{\lambda_3}{\lambda_l} \tag{7}$$

4. RESULTS AND ANALYSIS

The communication was performed on ChatGPT Dec 15 Version. The code generated were tested with the Stanford bunny point cloud (50 thousand points) in ply format in MATLAB 2022b version.

4.1 Surface normal estimation

The question asked was: *in MATLAB, generate a code to calculate the normals of a point cloud*. ChatGPT's response was:

To calculate the normals of a point cloud in MATLAB, you can use the function pcnormals, which is part of the Computer Vision System Toolbox. Here's an example of how you can use this function:

Load the point cloud data into MATLAB. For example:

```
ptCloud = pcread('pointCloud.ply');
```

Use the pcnormals function to calculate the normals of the point cloud. You can specify the number of nearest neighbors to use when calculating the normals, as well as the search radius. For example:

```
normals = pcnormals(ptCloud, 10);
```

The output of the pcnormals function is a matrix with the same number of rows as the input point cloud, and three columns for the x, y, and z components of the normal vector at each point. You can access the normals for a particular point by indexing into the matrix. For example:

```
normal = normals(100,:); % get the normal at
point 100
```

If you want to visualize the normals, you can use the pcshow function to display the point cloud, and then use the quiver3 function to plot the normals as arrows. For example:

```
pcshow(ptCloud);
hold on
quiver3(ptCloud.Location(:,1),
ptCloud.Location(:,2), ptCloud.Location(:,3),
normals(:,1), normals(:,2), normals(:,3));
hold off
```

I hope this helps! Let me know if you have any questions or need further assistance.

Based on the authors' prior knowledge of the programming language, this response was perfectly suited to the desired objective. In addition, ChatGPT provided an explanation of how the code works almost line by line, as well as references to the necessary library, an example output of a point and an output image (Figure 3).

In view of the success obtained, the next question was towards the variation of the number of nearest neighbours used for the calculation of normals: *in MATLAB, generate a code to calculate the normals of a point cloud based on 25 neighbours.* The answer given was a brief explanation of the code (which will not be reproduced below to save space), the Code 1, and an alternative Code 2.

https://doi.org/10.5194/isprs-annals-X-1-W1-2023-107-2023 | © Author(s) 2023. CC BY 4.0 License.

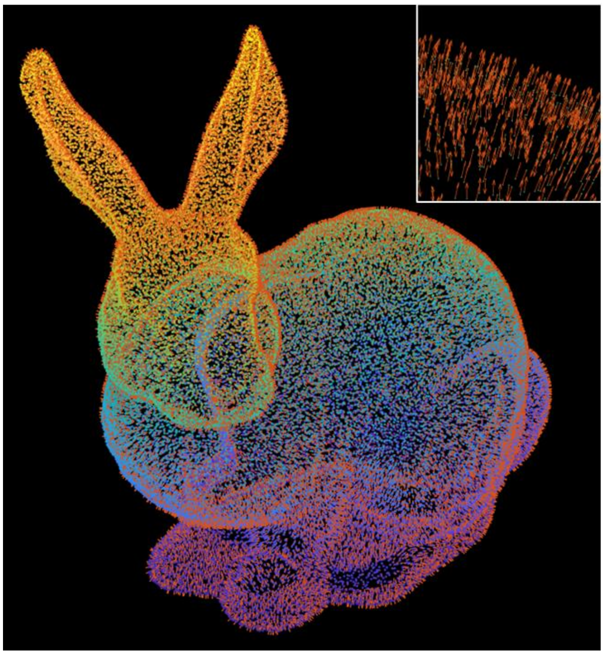

Figure 3. Image generated from Matlab code provided by ChatGPT of normal estimation and zoom in.

Code 1: Normal estimation k neighbours

```
% Load the point cloud
pc = pcread('pointCloud.ply');

% Extract the xyz coordinates of the point
cloud
xyz = pc.Location;

% Set the number of neighbours to use for
normal estimation
k = 25;

% Calculate the normals of the point cloud
normals = pcnormals(pc, k);
```

Code 2: Normal estimation (alternative)

```
% Load the point cloud
pc = pcread('pointCloud.ply');

% Extract the xyz coordinates of the point
cloud
xyz = pc.Location;

% Set the number of neighbours to use for
plane fitting
k = 25;

% Fit a plane to the points in the
neighbourhood of each point
[model, inlierIndices, outlierIndices] =
pcfitplane(pc, k);

% Extract the normal vector of the fitted
plane
normals = model.Normal;
```

In this case both codes have completely different outputs. Code 1 generated the surface normals point by point, while Code 2 obtained the normal to the plane that fit the entire point cloud. Both codes were fully functional and were commented line by line.

Since these codes produced no output, ChatGPT was asked for a specific output format: *in MATLAB, generate a code to calculate the normals of a point cloud based on 25 neighbours and save xyz points and normals to disk on a txt file*. The answer given was the following, in conjunction with Code 3.

To calculate the normals of a point cloud and save the xyz points and normals to a text file in MATLAB, you can use the pcnormals function as described in the previous answer and then use the dlmwrite function to write the xyz points and normals to a text file. Here is an example of how to do this:

Code 3: Normal estimation exporting data

```
% Load the point cloud
pc = pcread('pointCloud.ply');

% Extract the xyz coordinates of the point
cloud
xyz = pc.Location;

% Set the number of neighbours to use for
normal estimation
k = 25;

% Calculate the normals of the point cloud
normals = pcnormals(pc, k);

% Create a matrix with the xyz points and
normals
data = [xyz normals];

% Write the data to a text file
dlmwrite('pointCloudData.txt', data,
'delimiter', '\t');
```

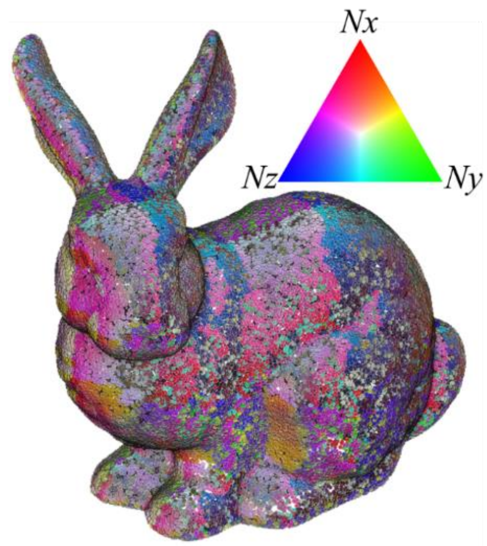

Figure 4. Point cloud coloured based on normals and imported in Cloud Compare.

This result can be considered a success. ChatGPT made references to the previous answer so as not to repeat the explanation, it added the explanation of the new function, and it generated a code that exported the point cloud on a txt file with a correct separation between columns. Bunny point cloud with estimated normal is visible in Figure 4).

4.2 Curvature

For the curvature calculation, the following question was asked: *in MATLAB, generate a code to calculate the point curvature of a point cloud based on 25 neighbours*. Unfortunately, the chat response (Code 4) referred to adding a curvature output to the calculation of normal, which was not correct, since this function does not support multiple outputs.

Code 4: Curvature (Attempt 1)

```
% Load the point cloud
pc = pcread('pointCloud.ply');

% Extract the xyz coordinates of the point
cloud
xyz = pc.Location;

% Set the number of neighbours to use for
normal estimation
k = 25;

% Calculate the normals and curvature of the
point cloud
[normals, curvature] = pcnormals(pc, k);
```

When it was indicated that 'pcnormals' function did not support two outputs (*Error using pcnormals. Too many output arguments*), ChatGPT proposed an alternative fitting the point cloud to a geometric model (Code 5):

I apologize for the error in the previous answer. It looks like the pcnormals function does not return the curvature of the point cloud when called with two output arguments.

Code 5: Curvature (Attempt 2)

```
% Load the point cloud
pc = pcread('pointCloud.ply');

% Extract the xyz coordinates of the point
cloud
xyz = pc.Location;

% Set the number of neighbors to use for plane
fitting
k = 25;

% Fit a plane to the points in the
neighbourhood of each point
[model, inlierIndices, outlierIndices] =
pcfitplane(pc, k);

% Extract the curvature of the fitted plane
curvature = model.Curvature;
```

However, that model does not support curvature field. By a new attempt, ChatGPT referenced a function that does not exist in MATLAB: *The surface normals can be calculated using the pcnormals function from the Computer Vision System Toolbox, and the curvature can be estimated using the pcurvature function*. And by indicating that the 'pccurvature' function does not exist, ChatGPT re-proposed Code 5.

Afterwards, an attempt was made to guide the GPT Chat by providing more information on how to calculate the curvature: *in MATLAB, generate a code to calculate the normals of a point cloud based on 25 neighbours. Then calculate the curvature based on variance of normal*. In the proposed response (Code 6), it can be seen how the indications were followed, however, the result was not as expected.

Code 6: Curvature (Attempt 3)

```
ptCloud = pcread('pointCloud.ply');
normals = pcnormals(ptCloud, 25);
variance = var(normals);
```

Finally, the authors decided to ask the ChatGPT what information it had on the point cloud curvature: *which is the point cloud curvature?* The following response was given:

In the context of point clouds, curvature refers to the degree of curvature or bend in a surface at a particular point. In other words, it is a measure of how much a surface deviates from being flat at a given point. In a point cloud, curvature can be calculated by analysing the shape and orientation of the points in the cloud and how they vary in relation to one another. There are several different methods for calculating curvature in a point cloud, including principal curvature, mean curvature, and Gaussian curvature. The specific method used may depend on the intended application and the type of data being analysed.

Based on the information provided (which was correct) and as there is no reference to the variance of surface normal to calculate the curvature, it was decided to ask for a code based on one of the methods mentioned: *in MATLAB, generate a code to calculate point cloud Gaussian curvature*. However, the answer (Code 7) again referred to a function that does not exist in MATLAB and the authors were decided to end this test without obtaining a correct code to calculate the curvature in point clouds.

Code 7: Curvature (Attempt 4)

```
% Load the point cloud data
ptCloud = pcread('pointCloud.ply');

% Fit a surface to the point cloud using
fitgmdist
gmModel = fitgmdist(ptCloud.Location, 2);

% Calculate the Gaussian curvature of the
surface at each point
K = gausscurvature(gmModel);
```

4.3 Eigenvalues

For the eigenvalues calculation, the following question was asked*: in MATLAB, generate a code to calculate the eigenvalues of a point cloud*. The answer referred to the Statistics and Machine Learning Toolbox with Code 8. The first thing to note was the change of input to a mat file. This was no problem, assuming that the point cloud is stored in a mat file. Next, Principal Component Analysis (PCA) was calculated, and the eigenvalues were indicated as the latent output. The code provided the eigenvalues of the point cloud understanding that eigenvalues are obtained from PCA and a correct output of the function PCA, but the code did not give the eigenvalues point by point.

Code 8: Eigenvalues (Attempt 1)

```
% Load the point cloud data
X = load('pointCloudData.mat');

% Perform PCA on the point cloud data
[coeff,score,latent] = pca(X);

% The latent variable contains the eigenvalues
of the point cloud data
eigenvalues = latent;
```

The following question tried to specify the output format a little more: *in MATLAB, generate a code to calculate the eigenvalues based on 25 nearest neighbours of a point cloud*. The answer is shown in Code 9. In this case, the code input is again a ply file. The kNN was calculated correctly, doing a point cloud neighbourhood search with itself. It is remarkable to mention that the code used a k = 26 neighbours correctly, since in a neighbourhood search of a point cloud with itself, the nearest neighbour of the point is the same point. Also, in the next line, the code corrected this fact by deleting the first column. However, the output of PCA was not correct, ignoring the correct information provided in Code 8. The code provided an output in the form of a bar chart (Figure 5).

Code 9: Eigenvalues (Attempt 2)

```
% Load the point cloud data
pc = pcread('pointCloud.ply');

% Find the indices of the 25 nearest neighbors
of each point
[nn, ~] = knnsearch(pc.Location, pc.Location,
'K', 26);
nn = nn(:, 2:end); % Remove the point itself
from the list of neighbors

% Perform PCA on the matrix of nearest
neighbor indices
[~, S, ~] = pca(nn);

% Visualize the eigenvalues
bar(diag(S))
xlabel('Principal Component')
ylabel('Eigenvalue')
```

Subsequent attempts did not give functional codes either. The chat proposed invented functions (pceigenvalues, pccovariance, pcdistancematrix) as well as examples that generate their own point cloud and used the eig function (Code 10). This code again produces an output of eigenvalues of the whole point cloud, so this experiment was terminated without success. However, it is interesting to observe how ChatGPT rendered correct assumptions such as PCA-dependent calculation of eigenvalues or covariance, and functions that calculate eigenvalues.

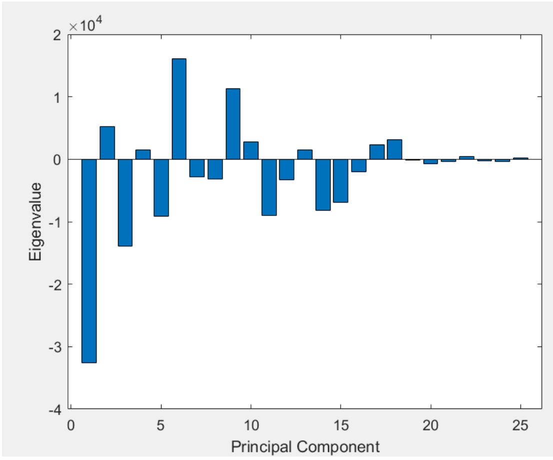

Figure 5. Output of the code for calculating eigenvalues (Attempt 2).

Code 10: Eigenvalues (Attempt 3)

```
% Define the data points of the point cloud as
a matrix, where each row represents a single
point and each column represents a dimension
of the data
points = [1 2 3; 4 5 6; 7 8 9; 10 11 12];

% Calculate the covariance matrix of the
points
covariance = cov(points);

% Calculate the eigenvalues and eigenvectors
of the covariance matrix
[eigenvectors, eigenvalues] = eig(covariance);

% The eigenvalues are stored in the diagonal
elements of the eigenvalues matrix
eigenvalues = diag(eigenvalues);

% Sort the eigenvalues in descending order
[eigenvalues, sortIndices] = sort(eigenvalues,
'descend');

% Reorder the eigenvectors in the same order
as the eigenvalues
eigenvectors = eigenvectors(:, sortIndices);
```

4.4 Eigenfeatures

Since these variables are dependent on the eigenvalues, for the calculation of linearity, planarity and scattering it was decided to propose a more complex question: *in MATLAB, generate a code to calculate point linearity, planarity and scatter in point clouds following the paper "Semantic point cloud interpretation based on optimal neighbourhoods, relevant features and efficient classifiers" of Martin Weinmann.* The first response from the ChatGPT was:

I'm sorry, but I am not able to generate code based on specific research papers or publications. I am a large language model trained by OpenAI and do not have access to specific research papers or publications. My training is based on a vast corpus of text data, and I am not able to browse the internet or access *specific research papers or publications. I can only provide general information and advice based on my training.*

However, a new attempt generated code (11) with the answer: *Here is a possible solution in MATLAB that follows the approach described in the paper "Semantic point cloud interpretation based on optimal neighbourhoods, relevant features and efficient classifiers" by Martin Weinmann.*

Code 11: Eigenfeatures (Attempt 1)

```
% load the point cloud
pc = pcread('pointCloud.ply');

% initialize arrays for storing the linearity,
planarity, and scatter of each point
linearity = zeros(pc.Count, 1);
planarity = zeros(pc.Count, 1);
scatter = zeros(pc.Count, 1);

% set the number of nearest neighbors to use
for normal estimation
numNeighbors = 20;

% calculate the surface normals of the points
using the "pcnormals" function
[normals, ~] =
pcnormals(pointCloud(pc.Location),
numNeighbors);

% loop through each point in the point cloud
for i = 1:pc.Count
    % extract the x, y, and z coordinates of
the point and its neighbors
    x = pc.Location(i,1);
    y = pc.Location(i,2);
    z = pc.Location(i,3);

    % extract the surface normal of the point
    normal = normals(i,:);

    % calculate the covariance matrix of the
point and its neighbors
    covariance = [x y z] \* [x y z]';

    % calculate the eigenvalues and
eigenvectors of the covariance matrix
    [eigenvectors, eigenvalues] =
eig(covariance);

    % store the linearity, planarity, and
scatter of the point
    linearity(i) = normal * eigenvectors(:,1);
    planarity(i) = normal * eigenvectors(:,2);
    scatter(i) = normal * eigenvectors(:,3);
end

% display the linearity, planarity, and
scatter of the points
disp(linearity);
disp(planarity);
disp(scatter);
```

Code 11 did not compile. The calculation of normals, covariances and eigenfeatures were not well implemented. Although it was the first time that ChatGPT proposed a loop to calculate the features point by point. In new attempts, Chat GPT proposed again the non-existent function pceigenvalues. Then it proposed the calculation of features as observed in Code 12. In both Code (11 and 12), ChatGPT was not able to calculate the eigenvalues well, which was expected based on the results of Section 4.3. But the Chat did not calculate the eigenfeatures correctly from the eigenvalues either, although he did establish some (wrong) relation between each eigenfeature and the eigenvalue, according Equation 5 to 7.

Code 12: Eigenfeatures (Attempt 2)

```
% Load the point cloud
pc = pcread('pointCloud.ply');

% Extract the xyz coordinates of the point
cloud
xyz = pc.Location;

% Set the number of neighbors to use for
eigenvalue calculation
k = 25;

% Calculate the eigenvalues of the point cloud
[eigenvalues, eigenvectors] =
pceigenvalues(pc, k);

% Calculate the linearity, planarity, and
scatter of the point cloud
linearity = eigenvalues(:,1) ./
eigenvalues(:,2);
planarity = eigenvalues(:,2) ./
eigenvalues(:,3);
scatter = eigenvalues(:,1) ./
eigenvalues(:,3);
```

5. DISCUSSION AND CONCLUSION

In this work the generation of ChatGPT codes for point cloud processing was evaluated. The codes were generated by the Chat automatically as answers to questions where the process to be performed was indicated (normal calculation, curvature, eigenvalues and eigenfeatures) and specific requirements such as number of neighbors or output format were added.

The evaluation of the codes was done by compiling and running the response in MATLAB. ChatGPT could generate short perfectly working code, usually based on very concrete functions. When more expert level code involving loop or index generation is required, the result was not satisfactory.

Provided functional codes can be useful for the first steps of people learning point cloud processing. Although examples of codes can also be easily found in help libraries. As for extensive codes, these require more advanced knowledge to correct them and make them functional. Many of which may be easier to write from scratch since they contain invented functions or incorrect equations/relationships.

Nevertheless, ChatGPT proves to be a useful tool for obtaining theoretical information on point cloud processing and simple code. Future work will focus on evaluating other widely used programming languages for point clouds, such as Python and C++, and develop complex codes step by step.

ACKNOWLEDGEMENTS

This work is supported by Xunta de Galicia (ED481B-2019-061), GAIN (ED431F 2022/08) and the ERDF ITMS 313011W988. This paper is a result of the project PID2021-123475OA-I00, funded by MCIN/AEI/10.13039/501100011033/ FEDER, UE.

REFERENCES

Che, E., Jung, J., Olsen, M., 2019. Object Recognition, Segmentation, and Classification of Mobile Laser Scanning Point Clouds: A State of the Art Review. Sensors 19, 810. https://doi.org/10.3390/s19040810

Dale, R., 2021. GPT-3: What's it good for? Nat Lang Eng 27, 113–118. https://doi.org/10.1017/S1351324920000601

Floridi, L., Chiriatti, M., 2020. GPT-3: Its Nature, Scope, Limits, and Consequences. Minds Mach (Dordr) 30, 681–694. https://doi.org/10.1007/s11023-020-09548-1

Gillioz, A., Casas, J., Mugellini, E., Khaled, O.A., 2020. Overview of the Transformer-based Models for NLP Tasks, in: 2020 15th Conference on Computer Science and Information Systems (FedCSIS). pp. 179–183. https://doi.org/10.15439/2020F20

Jolliffe, I.T., 2002. Principal component analysis for special types of data. Springer.

Li, Y., Ma, L., Zhong, Z., Liu, F., Chapman, M.A., Cao, D., Li, J., 2021. Deep Learning for LiDAR Point Clouds in Autonomous Driving: A Review. IEEE Trans Neural Netw Learn Syst 32, 3412–3432. https://doi.org/10.1109/TNNLS.2020.3015992

Liu, W., Sun, J., Li, W., Hu, T., Wang, P., 2019. Deep Learning on Point Clouds and Its Application: A Survey. Sensors 19, 4188. https://doi.org/10.3390/s19194188

OpenAI, 2022. ChatGPT: Optimizing Language [WWW Document]. URL Models for Dialogue, 2022. https://openai.com/blog/chatgpt/ (accessed 1.9.23).

Pauly, M., Gross, M., Kobbelt, L.P., 2002. Efficient simplification of point-sampled surfaces, in: IEEE Visualization, 2002. VIS 2002. pp. 163–170. https://doi.org/10.1109/VISUAL.2002.1183771

Wang, Q., Kim, M.-K., 2019. Applications of 3D point cloud data in the construction industry: A fifteen-year review from 2004 to 2018. Advanced Engineering Informatics 39, 306–319. https://doi.org/https://doi.org/10.1016/j.aei.2019.02.007

Weinmann, M., Jutzi, B., Hinz, S., Mallet, C., 2015. Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers. ISPRS Journal of Photogrammetry and Remote Sensing 105 (2015), 286–304. https://doi.org/10.1016/j.isprsjprs.2015.01.016

Xia, S., Chen, D., Wang, R., Li, J., Zhang, X., 2020. Geometric Primitives in LiDAR Point Clouds: A Review. IEEE J Sel Top Appl Earth Obs Remote Sens 13, 685–707. https://doi.org/10.1109/JSTARS.2020.2969119

Xie, Y., Tian, J., Zhu, X.X., 2020. Linking Points With Labels in 3D: A Review of Point Cloud Semantic Segmentation. IEEE Geosci Remote Sens Mag 8, 38–59. https://doi.org/10.1109/MGRS.2019.2937630

Yao Fu, Hao Peng, Tushar Shot, 2022. How does GPT Obtain its Ability? Tracing Emergent Abilities of Language Models to their Sources. [WWW Document]. URL https://yaofu.notion.site/How-does-GPT-Obtain-its-Ability-Tracing-Emergent-Abilities-of-Language-Models-to-their-Sources-b9a57ac0fcf74f30a1ab9e3e36fa1dc1 (accessed 1.9.23).

Zhang, M., Li, J., 2021. A commentary of GPT-3 in MIT Technology Review 2021. Fundamental Research 1, 831–833. https://doi.org/https://doi.org/10.1016/j.fmre.2021.11.011