Article

ChatGPT Translation of Program Code for Image Sketch Abstraction

Yulia Kumar [1,*], Zachary Gordon [1], Oluwatunmise Alabi [1], Jenny Li [1], Kathryn Leonard [2], Linda Ness [3] and Patricia Morreale [1]

[1] Department of Computer Science and Technology, Kean University, Union, NJ 07083, USA; gordonza@kean.edu (Z.G.); oalabi@kean.edu (O.A.); juli@kean.edu (J.L.); pmorreal@kean.edu (P.M.)
[2] Department of Computer Science, Occidental College, Los Angeles, CA 90041, USA; leonardk@oxy.edu
[3] Center for Discrete Mathematics and Theoretical Computer Science, Rutgers University, New Brunswick, NJ 08854, USA; ness.linda@gmail.com
* Correspondence: ykumar@kean.edu

Abstract: In this comprehensive study, a novel MATLAB to Python (M-to-PY) conversion process is showcased, specifically tailored for an intricate image skeletonization project involving fifteen MATLAB files and a large dataset. The central innovation of this research is the adept use of ChatGPT-4 as an AI assistant, pivotal in crafting a prototype M-to-PY converter. This converter's capabilities were thoroughly evaluated using a set of test cases generated by the Bard bot, ensuring a robust and effective tool. The culmination of this effort was the development of the Skeleton App, adept at image sketching and skeletonization. This live and publicly available app underscores the enormous potential of AI in enhancing the transition of scientific research from MATLAB to Python. The study highlights the blend of AI's computational prowess and human ingenuity in computational research, making significant strides in AI-assisted scientific exploration and tool development.

Keywords: MATLAB to Python (M-to-PY) converter; skeletonization; Skeleton App; ChatGPT; generative AI; Large Language Models (LLMs); machine learning; AI pair programming

check for updates

Citation: Kumar, Y.; Gordon, Z.; Alabi, O.; Li, J.; Leonard, K.; Ness, L.; Morreale, P. ChatGPT Translation of Program Code for Image Sketch Abstraction. *Appl. Sci.* **2024**, *14*, 992. https://doi.org/10.3390/app14030992

Academic Editor: Ting Xiao

Received: 29 October 2023
Revised: 12 January 2024
Accepted: 16 January 2024
Published: 24 January 2024

1. Introduction

Transitioning from MATLAB (R2023b prerelease) to Python (version 3.10), an M-to-PY conversion, is a complex undertaking that typically necessitates a deep understanding of both programming languages. However, this study introduces a systematic approach to this conversion that can be executed even without such extensive knowledge. The outcomes of each step are compared with those of existing tools and several previous unsuccessful attempts, including those involving earlier versions of the ChatGPT bot, based on test cases generated by ChatGPT-4, Bing, and Bard. A prototype converter, underpinned by the OpenAI API [1], is introduced. This converter leverages the sentiment analysis capabilities of the Large Language Model (LLM) to understand the context and semantics of the code. This is a significant advancement over other known programming language converters such as Libra, SMOP, Mat2Py, and the NumPy, Oct2Py, and SciPy Python packages and libraries. These tools focus solely on code syntax and may suffice for simple functions or sequential code, but they fall short when confronted with more complex code structures. The study seeks to address the following research questions:

RQ1: Can LLMs be effectively used as AI pair programmers in MATLAB-to-Python conversion for complex projects?

RQ2: How do the results of conversions performed by LLMs compare with existing language-to-language converter tools regarding accuracy and efficiency?

RQ3: Can the process of MATLAB-to-Python conversion be automated using the OpenAI API, and what are the potential challenges and solutions in this process?

RQ4: How feasible is it to make a Skeleton App with the help of ChatGPT once the M-to-PY translation is completed?

This study focused on the MATLAB one-step compact skeletonization code developed by Kathryn Leonard et al. [2]. Once the team succeeded in making the original code work, the following results were obtained: see Figure 1.

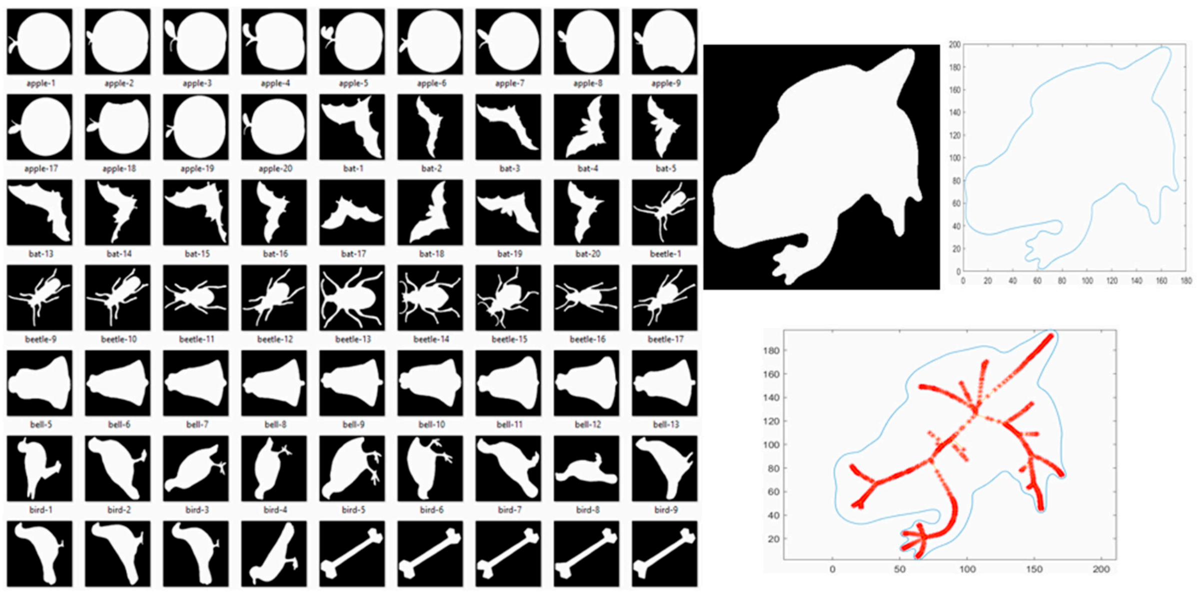


Figure 1. Blum skeletonization produced by MATLAB: a snippet of the contour dataset (**left**), frog's contour and sketch (**top right**), and skeleton (**bottom right**).

Figure 1 is a segment of the initial dataset showcasing Blum skeletonization. The figure displays a snippet of the contour dataset (top left) and a detailed contour of a frog (top right), followed by the sketch and the resulting skeleton (bottom left and right, respectively).

The dataset, partially illustrated in Figure 1, comprises 1404 contour images, including the frog image. This dataset was explicitly selected for its diversity and suitability for the skeletonization processes. The images, characterized by their clear boundary definitions, are ideal for demonstrating the efficacy of the MATLAB to Python conversion and the subsequent skeletonization technique.

The original MATLAB code, tailored for this dataset, faced several challenges due to inconsistencies in software tools and code versions. It employed MATLAB-specific functions, such as "inpolygon()", and incorporated a custom Dijkstra algorithm, as detailed in Adeel Javaid's implementation [3]. Optimized for MATLAB's data structures, the code used complex numbers to represent two-dimensional arrays, posing additional challenges for the junior developer team during the debugging phase before applying LLMs.

The MATLAB code's primary purpose was to identify boundary coordinates and calculate medial axis points for each image, known as image sketch abstraction or object skeletonization. The resultant skeleton, overlaid on each boundary image, signifies the object's center of gravity. This aspect of skeletonization is crucial for motion tracking in AI research, a topic further explored later in this paper. The conversion of this project to Python is expected to advance the field of 3D motion detection with AI transformers, presenting a valuable resource for other researchers in this area.

The case study mainly used ChatGPT-4 as an AI Pair Programmer tool throughout all conversion processes with the minor use of Bard and Bing chatbots. It involved chatting with the bot about the best strategies, asking it to browse the web for answers (the feature is currently shut down for ChatGPT-4), writing, debugging, cleaning the code, and creating test cases. The OpenAI API and ChatGPT-4 models were then used for live M-to-PY conversion, available through the Skeleton App. The DALL-E Open AI tool generated the app logo, which generates images based on the text prompts and is available to be used programmatically. The logo's color theme and font family were then detected by the Open AI Code Interpreter—a file analyzer [4]—and for consistency for all app layouts. Therefore, this research leverages the capabilities of LLMs to facilitate the M-to-PY conversion and creates a Flask Skeleton App, creating a new tool in hybrid web development and AI.

2. Related Work

The initiation of this study predates the public release of ChatGPT, and several converter tools, including OMPC [5] and Mat2Py, were initially utilized with varying degrees of success. The authors were aware of other existing tools and packages such as pymatlab, Python-Matlab wormholes, Python-Matlab bridge, pymat2, MatPy, and PyMat Python packages, among others. Efforts were made to find a tool that employed deep learning for such conversion, with Meta's TransCoder [6] being a notable example. However, this tool was primarily focused on conversions between COBOL to C++ and C++ to Python. After an intensive search and trial period, the tool matlab2python was selected as the most suitable. This tool, which relies on the more renowned converter SMOP with some modifications, can accommodate the translation of MATLAB code lines and entire files.

The application of Large Language Models (LLMs), such as ChatGPT, across various fields has been the subject of extensive exploration in recent studies. These models have demonstrated promising results in diverse areas, including engineering, education, and science, thereby showcasing their versatility and potential. In engineering, Koziolek et al. [7] investigated the use of ChatGPT for generating control logic for programmable logic controllers and distributed control systems, finding that ChatGPT could generate code as accurately as humans. Similarly, Ran Li et al. [8] leveraged ChatGPT for power system programming tasks, demonstrating that it could generate code as efficiently as human experts. Meng-Lin Tsai et al. [9] explored the use of LLMs in chemical engineering education, finding that LLMs could generate core course problem models that were more engaging and effective than traditional problem models. This suggests that LLMs, such as ChatGPT, could play a crucial role in enhancing the learning experience in engineering education. In the context of scientific coding, Cory Merow et al. [10] found that AI chatbots could boost scientific coding by assisting scientists in writing code more quickly and accurately. Yulia Kumar et al. [11] proposed a comprehensive testing framework for AI linguistic systems using ChatGPT (version 4) as an AI pair programmer. They found that ChatGPT could generate test cases more effectively than human experts, indicating its potential in the M-to-PY conversion for complex projects such as image skeletonization. This conversion process could potentially be automated using the OpenAI API. Other studies have focused on the potential impact of AI language models, such as ChatGPT, on various professions. Brett Maurer [12] discussed how AI chatbots, such as ChatGPT, could revolutionize the geotechnical engineering profession by automating tasks, providing advice, and answering questions. In a different vein, Nuno Crokidakis et al. [13] used ChatGPT to chat about complex systems and found that it could generate informative and engaging responses. Sasha Nikolic et al. [14] conducted a multidisciplinary and multi-institutional benchmark and analysis of ChatGPT to investigate assessment integrity in engineering education. These studies found that ChatGPT could generate code as accurately as code created by software developers, making it hard to distinguish between the two.

While developing the M-to-PY converter prototype, the researchers conducted performance validation against the converters, the main competitors of the proposed tool: AI Code Translator Vercel [15], CodeConvert AI [16], and ThereIsAnAIForThat. These tools also use the ChatGPT-4 model for code translation. However, this work is distinct as the researchers focus specifically on Image Sketch Abstraction (skeletonization) and translating large custom projects, while these tools translate a variety of languages utilizing the OpenAI API and can be considered as another language converter used for line-to-line or small snippet conversions.

Several recent studies have explored the use of skeletonization in various applications, demonstrating its potential in diverse fields. Jun Ma et al. [17] proposed a noise-against-skeleton extraction framework and applied it to hand gesture recognition, demonstrating the potential of skeletonization in the field of human-computer interaction. Similarly, Taohan Wang and Yamakawa Yuji [18] developed an edge-supervised linear object skeletonization method for high-speed cameras, which could be particularly useful in real-time tracking and analysis of fast-moving objects. The use of skeletonization in human activity recognition has also been extensively studied. Mayank Lovanshi and Tiwari Vivek [19] used the human skeleton pose and spatio-temporal features for activity recognition, employing a spatio-temporal graph convolutional network (ST-GCN) for this purpose. Atiya Usmani et al. [20] also focused on human activity recognition, but they used deep recurrent neural networks (RNNs) to analyze skeleton joint trajectories. These studies highlight the potential of skeletonization and related techniques in understanding and interpreting human activities, which could have significant implications for fields such as surveillance, healthcare, and human-computer interaction. Considering these studies, this project will also incorporate a Skeleton app, which will leverage the capabilities of the M-to-PY converter and the ChatGPT-4 AI Pair Programmer. This app will further demonstrate the versatility and potential of these technologies in practical and research applications.

3. Problem Setting and the Skeleton App

The M-to-PY conversion of a custom image skeletonization project is a critical component of the authors' broader AI research agenda, which aims to implement 3D motion detection, as depicted in Figure 2.

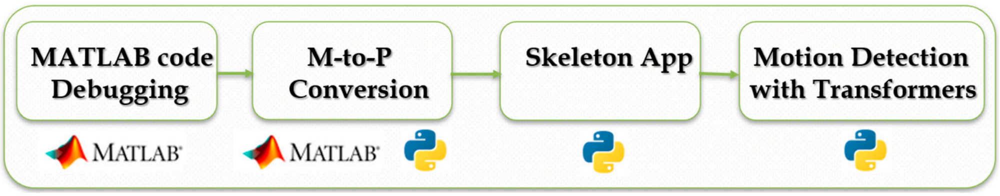


Figure 2. The project plan with M-to-PY translation and the Skeleton App are the middle tiers of the study.

Figure 2 shows the project plan, consisting of four main steps with an additional implementation step between tool usage. As such, the MATLAB code Debugging stage required obtaining the initial code from the authors, getting a MATLAB license, and setting up the project on a computational-intensive Windows Alienware machine to which several research team members could connect remotely. Before using Large Language Model (LLM) tools, such as ChatGPT-4, Bing, or Bard, for code comprehension and debugging, the researchers had to manually add comments to almost every line of code to comprehend the logic behind the original MATLAB code. The codebase consisted of 15 files; their content can be observed at-a-Glance from the word cloud in Figure 3.

Figure 3 shows such words as boundary, points, and medialData that prevail in the code as the initial emphasis was on image sketch abstraction from the center of balance, a pivotal aspect of this research that lays the groundwork for enhancing motion detection and tracking. The process commences by calculating the target object's center of gravity, followed by creating a skeleton of the object.

The initial M-to-PY conversion took an extended amount of time, with several junior developers attempting to make the project work using various converters with little to no success. Then, the conversion made significant progress with the release of the early version of ChatGPT, ChatGPT-3, and leaped toward completion once the team started to use LLMs and tools such as GitHub Copilot. As the required solution involves delineating the outline of the target object and constructing the object's skeleton, it was essential to test a variety of images to understand the capacities to which the skeleton can be successfully generated. The team verified that all 1404 images in the initial dataset were processed correctly and began testing newly acquired images outside the dataset. This step was thus far the hardest. The code output was a folder of image files of the tested objects, each with an accurate outline, skeleton, and center of gravity map on a coordinate plane. Various environments were explored, such as Anaconda, Visual Studio Code, and Google Colab, each bringing some progress to the M-to-PY conversion, but the main contribution to the project was made by ChatGPT-4 as a pair programmer to generate the Skeleton app (RQ1).

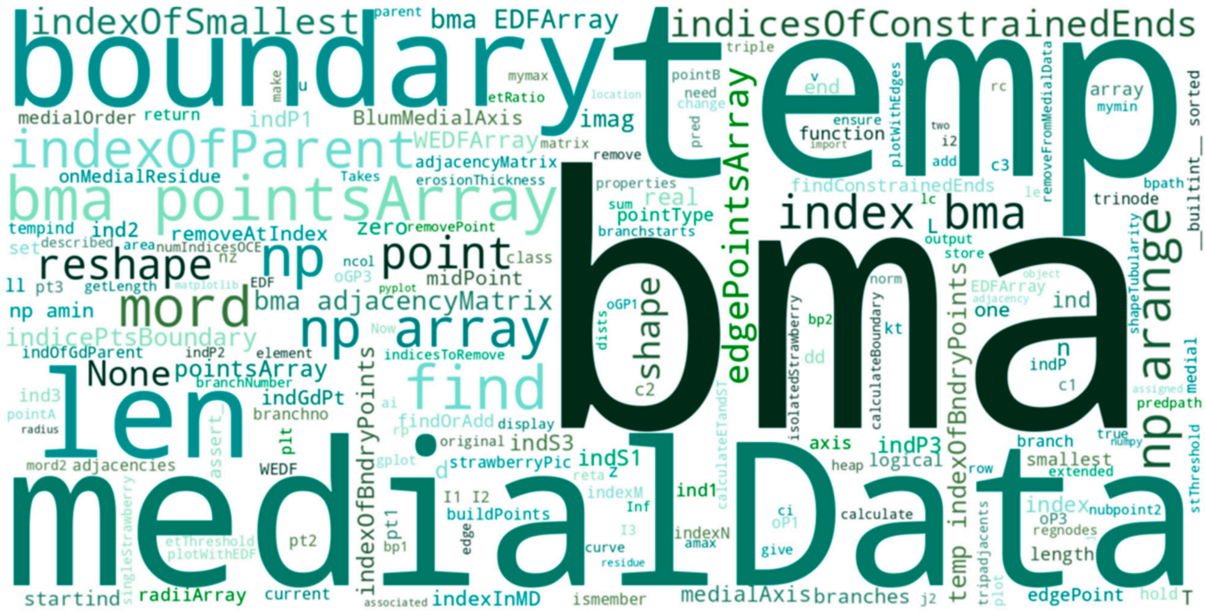
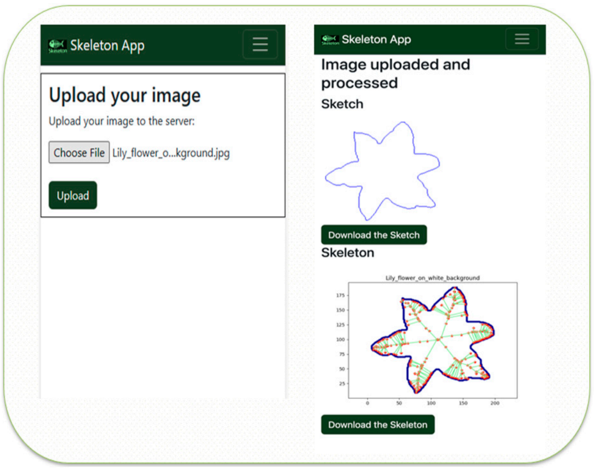
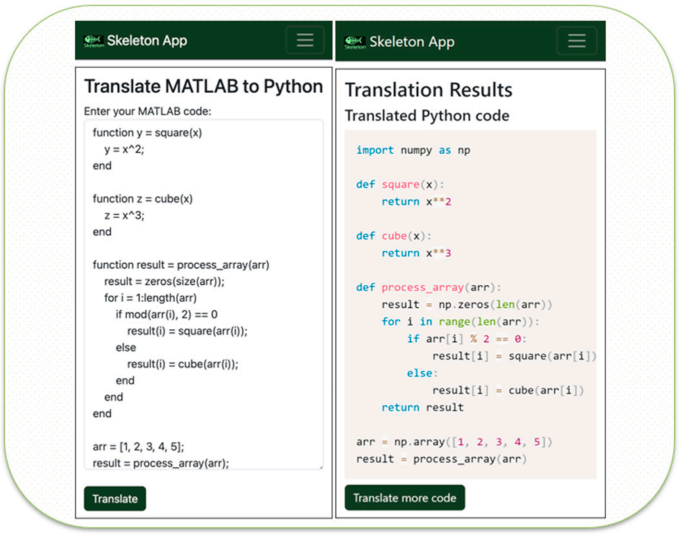


Figure 3. Sentiment analysis of the MATLAB code.

The Skeleton app [21] is a Flask web app written in Python. It integrates the Skeleton Generator page, where the users can upload the initial file. They are then redirected to the Result page, where they can download both sketches and the skeleton of their object; the M-to-PY converter resides on the app's M-to-PY page. The Sketches History is an additional page that displays the history of the previously created sketches. The results of previous conversions are currently not displayed, though stored in the file system, as the researchers are undecided about long-term data storage. Figure 4 demonstrates the Home and Result web pages (left) and the M-to-PY translation page of the app (right).


Skeleton App
Upload your image
Upload your image to the server:
Choose File
Lily_flower_o...kground.jpg
Upload
Skeleton App
Image uploaded and processed
Sketch
Download the Sketch
Skeleton
Lily_flower_on_white_background
Download the Skeleton



```
function y = square(x)
    y = x^2;
end

function z = cube(x)
    z = x^3;
end

function result = process_array(arr)
    result = zeros(size(arr));
    for i = 1:length(arr)
        if mod(arr(i), 2) == 0
            result(i) = square(arr(i));
        else
            result(i) = cube(arr(i));
        end
    end
end

arr = [1, 2, 3, 4, 5];
result = process_array(arr);
```

```python
import numpy as np

def square(x):
    return x**2

def cube(x):
    return x**3

def process_array(arr):
    result = np.zeros(len(arr))
    for i in range(len(arr)):
        if arr[i] % 2 == 0:
            result[i] = square(arr[i])
        else:
            result[i] = cube(arr[i])
    return result

arr = np.array([1, 2, 3, 4, 5])
result = process_array(arr)
```

Figure 4. Home and Result web pages of the Skeleton app (**left**), and M-to-PY translation page (**right**).

The translation, presented in Figure 4 (right), is performed by the OpenAI API and ChatGPT-4 models (RQ3, RQ4). As can be seen from Figure 4, the Skeleton app is responsive, simple, and user-friendly. The only action the user needs to take is to upload/download the images to obtain sketches and the skeleton. The Skeleton app architecture can be observed in Figure 5. It is novice, unique, and can scale quickly if deployed live.

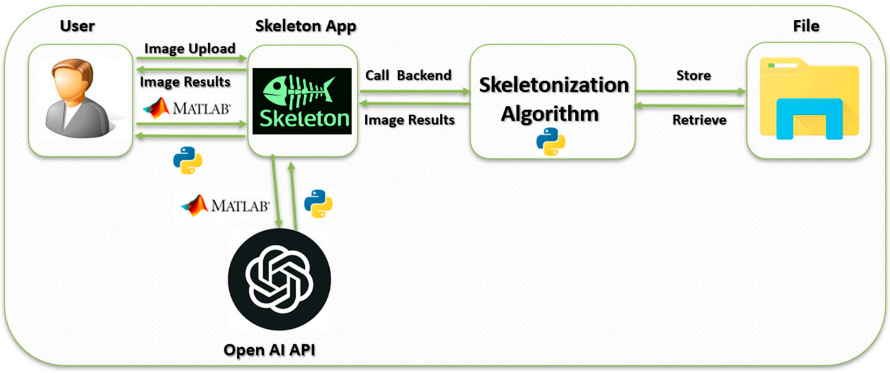


Figure 5. The Skeleton app architecture.

The app might be released in an app store in the future. At this point, the researchers utilize a file system to store images and code snippets to facilitate future expansion. As the App usage grows, it might be essential to connect to a database or a cloud service, or both. Currently, the app only serves research purposes, and data storage is not an issue.

Currently, the last stage of the project is in progress, and the researchers are planning to publish it as a separate study unrelated to the M-to-PY conversion and the Skeleton app. There has recently been substantial progress in motion detection, and it is important to understand the most relevant approaches and how the presented approach will differ from the others. One of the ongoing investigations is to use and validate existing images from 2D or 3D videos to test out and polish the current Skeleton app.

Figure 6 shows one of the testing outcomes. By pinpointing a subject's center of gravity and creating a skeleton and outline for the object, the researchers eventually anticipate significant enhancements in motion-tracking accuracy compared to existing methodologies.

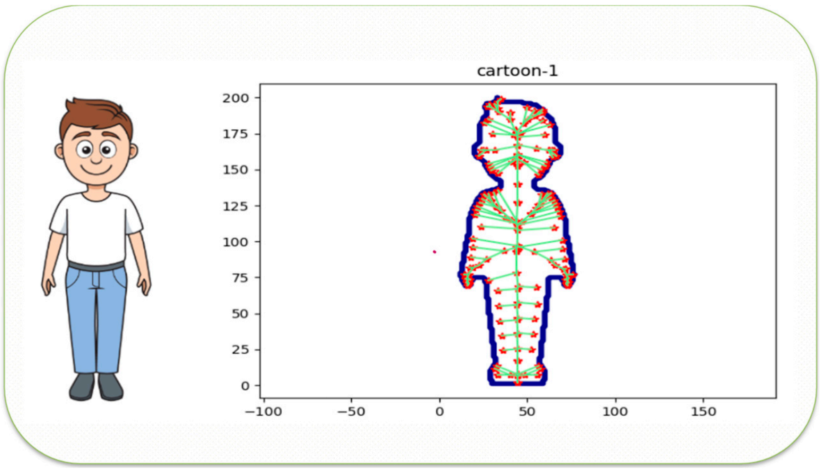


Figure 6. The cartoon person and the generated skeleton.

4. Methodology of the M-to-PY Conversion

4.1. The M-to-PY Methodology

The methodology for the M-to-PY conversion for a complex project, such as a One-step Skeletonization, is an iterative process, that can be reproduced step-by-step. The key steps are presented below:

- Select LLM(s) to work with;
- Understand the original MATLAB code *with the help of LLM(s)*;
- Identify conversion challenges;
- Develop the M-to-PY converter prototype *with the help of LLM(s)*;
- Develop an associated App (optional);
- Test and refine the converter *using test cases generated by LLM(s)*;
- Evaluate and learn from the result;
- Compare the results with existing tools.

Selecting Large Language Models (LLMs) is a critical step in this methodology. The researchers mainly worked with several versions of ChatGPT, Bard, and Bing chatbots and with the GitHub Copilot tool. Considering the expected rise of new LLMs, it is important to be aware of the latest developments in the field. The LLMs should be chosen based on their ability to understand the context and semantics of the code, a significant advancement over other known converters that focus solely on code syntax.

Understanding the Original MATLAB Code is the foundational step where the team comprehensively understands the code they work with. This is crucial as it identifies the key components of the code that need to be translated and the potential challenges that might arise during the conversion process. It is suggested to either ask the LLM(s) of choice to explain the code, its functions, modules, and classes, or add comments to each line of code (what the researchers had to do manually) or chat with the bot, for example, with the one embedded in the GitHub Copilot.

Identifying Conversion Challenges involves understanding the differences between MATLAB and Python syntax, identifying MATLAB-specific functions that might not have direct equivalents in Python, and considering the complexity of the mathematical concepts used in the code. This step is instrumental but equally important as it might require most human intervention.

The development of the M-to-PY Converter prototype can be significantly sped up with the help of LLMs. The research team manually worked with the chatbots on the ideas and layout drafts, then developed a prototype converter using the OpenAI API. As previously mentioned, the DALL-E 2 text-to-image generation tool created the Skeleton app logo. Key points to consider are simplicity, accessibility, responsiveness, equitability, and inclusivity.

Developing an associated app is optional as it depends on the team's goals. For the researchers, developing the Skeleton app in parallel with the development of the converter was natural because of their skilled web programming. The Skeleton app was created along the side of the converter to demonstrate the practical application of M-to-PY conversion in the field of image sketch abstraction for 3D motion detection.

Testing and refining the converter and the app is one of the critical steps in the process. The researchers combined several LLMs to complete this step. First, ChatGPT-4, Bard, and Bing were instructed to generate high-view-point test cases. Bard was chosen as the best creator of those test scenarios, and then ChatGPT-4 produced the actual code to test. Bard proposed the following test cases. Their complete AI-generated code can be found online [22]:

(1) Complex Data Structure—Linked List;
(2) Complex Algorithm—Quick Sort;
(3) Complex Library—NumPy;
(4) Error-Prone Code—Recursion;

(5) Multithreading;
(6) Modular code.

As mentioned above, diverse and comprehensive testing parameters were employed in this study. These parameters encompass a broad spectrum of coding complexities and challenges, including complex data structures, intricate algorithms, advanced libraries, recursion handling, multithreading capabilities, and modular code [22].

While it seems that Bing was an outsider in this process, its ability to browse the internet was very important in this study. Obtaining the latest AI and development news and tools is critical to be compatible and relevant in this research study. The Skeleton app was straightforward in its ability to create sketches and the skeleton. The accuracy was evaluated based on the accuracy of generated sketches, skeletons, and center of gravity maps on a coordinate plane for each image.

Further evaluation of these results is ongoing, with attempts to further improve and validate accuracy and efficiency before its final release to the research community as a platform for both code conversion and image sketch abstraction. The converter will facilitate scientific researchers' programs, often written in MATLAB, to utilize the latest AI algorithms, most often in Python. Sketch abstraction will contribute to the algorithms for 3D motion detection.

The step of Comparing with Existing Tools includes performance evaluation. The performance of the prototype converter was compared with existing language-to-language converter tools, including the AI Code Translator Vercel, CodeConvert AI, and OMPC. These tools also use ChatGPT-4 for code translation but focus on line-to-line or small snippet conversions, unlike this project, which focuses on converting large custom projects.

As can be seen from Table 1, the M-to-PY converter outperformed all its competitors, including the Vercel converter, which also uses ChatGPT-4.

Table 1. Pair Converter Tools Comparison (RQ2).

Converter Tool	**Test Case Number**					
	1	**2**	**3**	**4**	**5**	**6**
M-to-PY	✓	✓	✓	✓	✓	✓
Vercel	✓	✓	✓	✓	✓	×
CodeConvert AI	✓	✓	✓	✓	×	×
OMPC	×	×	×	×	×	×

The methodology for developing the Skeleton App, which is a part of the M-to-PY conversion project, is also an iterative process. It is a by-product of the converter that was used to test the effectiveness, accuracy, and performance of the generated converter. The workflow of the Skeleton App generation includes eight key steps:

- Choosing programming language and database;
- Designing the Skeleton App GUI;
- Developing the Skeleton App prototype;
- Testing the Skeleton App;
- Refining the Skeleton App;
- Comparing with existing tools;
- Creating test cases;
- Evaluating the results.

As OpenAI API provides QuickStart tutorials only for React.js JavaScript (version 18.0) and Python Flask web apps (version 2.3.1) [1], the researchers chose the second one to be consistent with the M-to-PY converter. With the novel paradigm of AI code generation, software engineering and development can shift its focus from actual coding to extensive testing and maintenance.

4.2. Subsection Experiments with LLMs

The empirical validation of the effectiveness of LLMs in the one-step skeletonization conversion project from MATLAB to Python (M-to-PY) comprises three critical components: AI pair programming, the generation of test cases by LLMs, and the capacity of LLMs for programming language-to-language translation. After integrating LLMs into the study, rigorous testing and benchmarking against existing language-to-language converters and other AI programming tools were conducted. The challenge lies in the fact that few intelligent models, such as ChatGPT-4, can currently be used as a backend for the M-to-PY converter, similarly as its main competitors use the same exact model. The distinguishing difference of this project is the use of a complex custom object skeletonization code that the researchers used as a test case for the converter and the byproduct of this research.

4.3. Pair Programming with LLMs

The application of pair programming with AI was implemented in the ongoing "Image abstraction from the center of balance of motion detection" project, as the case study by Yulia Kumar et al. [11]. The original code, written in MATLAB and provided by its authors, had a history of conversion attempts that started before the release of ChatGPT. After several initial attempts with M-to-PY converters performed by junior developers, about 30% of the initial MATLAB code was translated and tested. These were mainly static functions unrelated to the two objects used in the code. A comparison of existing tools demonstrated that in comparison with several other converters such as Libra, SMOP, Mat2Py, and the NumPy, Oct2Py, and SciPy Python packages and libraries, the tool matlab2python produced the best results. At that time, the results of the whole project execution were obscured, and its success was uncertain. It was stated that such a complex project could not be translated by simple, unintelligent tools by junior developers without previous knowledge in MATLAB and with minimum knowledge in Python, which was later established as a condition to validate LLM's effectiveness in the process. The releases of ChatGPT-3, ChatGPT-3.5, and ChatGPT-4 all served as trampolines to move the conversion further.

With the first release, both MATLAB and Python environments were set up, and researchers started asking ChatGPT-3 to translate the code through its chat capacity. It was impossible to provide all the code simultaneously due to ChatGPT-3 limitations on both input and output. As the code was object-oriented and consisted of both static and instance functions that belong to their classes, it was very difficult to split it into chunks that could be easily fed into the model one by one and then, without much difficulty, could be assembled back. Python code upgrades to the latest version of the language itself and its libraries added level of complexity. The code was translated in small chunks, and a human copilot had to perform most of the debugging and troubleshooting. The project saw some progress with the early ChatGPT-3 model but the whole code still did not run at once. Situations when not all input was accepted, or part of the code was sent in response, were very disappointing. The attempt at auto-debugging was unsuccessful. The editor, Visual Studio Code (version 1.81), was used with little success for human debugging. A significant problem was situations when the shape or dimensions of data structures that were returned from a function in Python (aka arrays) did not match the MATLAB code. Many cases of array vs. list data structures were encountered where some functionality that could work with the array was not working with lists and vice versa. At the same time, this AI pair programming experience was very useful for team members to understand the LLM capabilities, improve prompt engineering skills, and understand the source code better.

The prominent test cases of the conducted experiments are in Table 2.

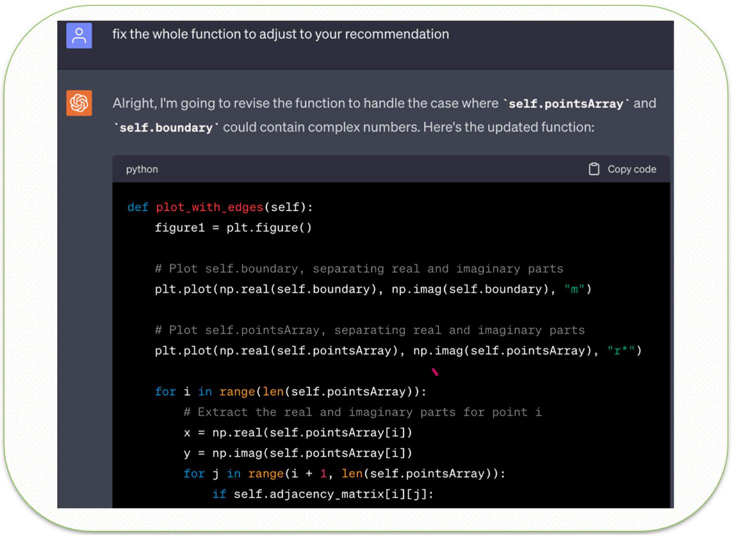
Table 2. Pair Programming with ChatGPT-3.5.

#	Testing Parmeter	Outcome	Testing Notes
1	Feeding whole text at once	Failure	ChatGPT did not allow more than 1000 tokens
2	Feeding MATLAB code snippets in parts	Failure	Many errors as chat generate not only different variable names but assume and produce different types and shapes of data structures
3	Running whole code	Failure	Did not work hard to conclude as to why the source code consists of several classes, modules, and many functions. Only one starting main script runs calling other modules, creating objects, etc.
4	Possible workarounds	Failure	Not many available tools, converters did not work, and manually debugging required time and expertise with MATLAB

The release of ChatGPT-3.5 gave a new breath to the project, and visible progress was made. It should be noted that it was worth it to rerun the code already generated by ChatGPT-3 after the new version of the bot came out. The goal at this point was not to translate the code as soon as possible but to test and validate the AI Pair programming experience with ChatGPT-3.5, comparing it with the previous trials and tools. The overall quality of the code generation from version to version increased.

The release of ChatGPT-4 was the event that made the study successful. The model's "understanding" of the code was significantly improved. It can be compared to the transition from working with an entry-level developer as a pair programmer whom you must lead to an advanced junior or early senior developer who might lead you. The team even invented the terminology of a human copilot as the model served as a team lead. A significant bottleneck was the commercial nature of the model and its usage caps. ChatGPT-4 suggested taking a break and returning after several hours or using the default model, ChatGPT-3.5. For the purity of the test trials, taking breaks was preferred in this case. The breakthrough happened when the produced skeleton was seen in the output. ChatGPT could not only explain the code and the errors but also get working results when making the skeletons. At the same time, it was still ChatGPT-4 who wrote the code.

As can be seen from Figure 7, it is the ChatGPT-4 model that produced the complete code for a function. The results of the successful run can be observed in Figure 8.


fix the whole function to adjust to your recommendation

Alright, I'm going to revise the function to handle the case where `self.pointsArray` and `self.boundary` could contain complex numbers. Here's the updated function:

```python
def plot_with_edges(self):
    figure1 = plt.figure()

    # Plot self.boundary, separating real and imaginary parts
    plt.plot(np.real(self.boundary), np.imag(self.boundary), "m")

    # Plot self.pointsArray, separating real and imaginary parts
    plt.plot(np.real(self.pointsArray), np.imag(self.pointsArray), "r*")

    for i in range(len(self.pointsArray)):
        # Extract the real and imaginary parts for point i
        x = np.real(self.pointsArray[i])
        y = np.imag(self.pointsArray[i])
        for j in range(i + 1, len(self.pointsArray)):
            if self.adjacency_matrix[i][j]:
```

Figure 7. Code revision by ChatGPT.

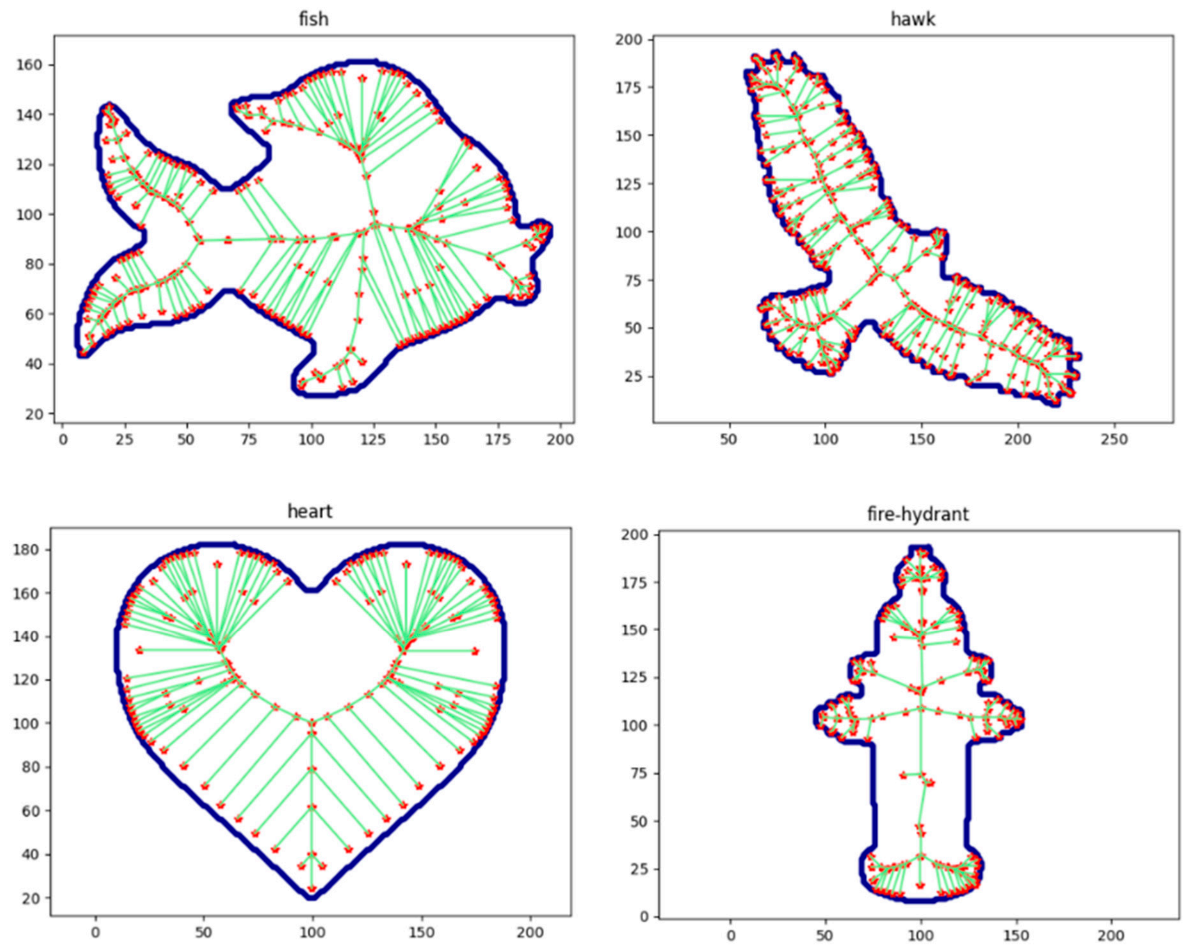


Figure 8. The Python code produces skeleton images.

4.4. LLMs Test Generation

Generating test cases with LLMs was another crucial aspect of the study. The team used ChatGPT-4 to generate test cases for the MATLAB to Python (M-to-PY) conversion project. The model was asked to generate test cases based on the test scenarios provided by Bard. Interestingly, the Bard bot happily generated the test scenarios evaluated as more solid than ChatGPT's ones, but the model refused to provide the actual code snippets to test. This study's finding is that this step was accomplished by combining the results from two various LLMs. The generated test cases were then used to validate the functionality of the converted Python code. This approach proved highly effective, as it allowed for comprehensive code testing without requiring manual test case creation. The Bard LLM was able to generate a wide variety of test cases, covering different scenarios and edge cases that might not have been considered otherwise. This saved time and improved the quality of the testing process, leading to more robust and reliable Python code.

4.5. M-to-PY Conversion with LLMs

The M-to-PY conversion was the core of the study, and LLMs played a pivotal role in this process. The team used ChatGPT-4 to translate the MATLAB code into Python and eventually added it to the backend of the Skeleton App. It was proven that LLMs could understand the logic and structure of MATLAB code and translate it into Python accurately. The translated code was then tested using the test cases generated by both Bard and ChatGPT-4. The LLM was not only able to translate the code but also provided explanations and insights into its functionality and added comments to it. This was particularly helpful in understanding the complex parts of the code. The LLM was also able to identify and correct any errors in the MATLAB code, improving the overall quality of the Python code. The code produced by the LLM was functional, accurate, and efficient. The LLM could translate complex MATLAB code into Python, a previously challenging task for other tools and methods. This success demonstrates the potential of LLMs in programming language-to-language translation and opens new possibilities for future research and applications. The translation process was not without its challenges. The researchers encountered issues with the size of the code snippets that could be fed to the bot, differences in the data structures assumed by the bot, and inaccuracies in the translated code. However, they observed an improved quality of the translated code with each new version of ChatGPT.

The authors developed a Skeleton App as a part of this project. This app takes an image as input and returns its sketches and a skeleton. The app was developed using the Flask framework for the backend and Bootstrap for the front end, with the assistance of ChatGPT-4′s Code Interpreter and the DALL-E text-to-image generator. The app demonstrates the practical application of the M-to-PY conversion process.

Throughout the project, the authors kept track of the progress and documented all findings. They then compared the performance of their M-to-PY converter with existing tools and created complex test cases to evaluate its performance.

They also evaluated the performance of the Skeleton App using a variety of metrics, including the accuracy of the skeletonization, the speed of the app, and the quality of the user interface. On average, the skeleton generation took 4.32 milliseconds.

This research provides valuable insights into the capabilities and limitations of LLMs. It demonstrates their potential in facilitating the development of practical apps. The findings will contribute to the ongoing research in this field and provide a foundation for future studies.

5. Motion Detection with ChatGPT-4

Once the MATLAB skeleton code was translated to Python, and the M-to-PY converter, together with the Skeleton app, was fully tested and deployed [21,22], it was decided to create a case study of conducting motion detection supervised by ChatGPT. Following the logic of the original Skeleton project [2], the ChatGPT-4 model was set to do an Advanced Data Analysis, last updated on 25 September 2023. It was asked to walk the researcher through the process of motion detection based on the information it contains. The chats with the chatbots and generated code outcomes were all recorded and can be found publicly [23,24]. More examples can be provided on demand.

Table 3 explains the applications, pros, and cons of creating Object Outlines and Skeletons in such a study.

Table 3. Object Outlines vs. Skeletons for Motion Detection.

Criteria	Object Outlines	Skeletons
Representation	The complete boundary of the object	Simplified, centerline representation of the object
Pros	A clear understanding of shape and orientation Easy calculation of features such as area and perimeter	Simplified analysis of complex shapes Less sensitive to boundary noise Useful for analyzing the movement of object parts
Cons	Sensitive to noise and small boundary variations	Loses information about object's width and shape
Application to Motion Detection	Better for understanding overall object movement and shape changes	Better for analyzing the movement of specific object parts

As can be seen from Table 3. Both object representation types are used for motion detection and contribute to the results in various ways.

A high-view logic of the code created under the guidance of ChatGPT can be seen below (Algorithm 1):

Algorithm 1: High-view pseudocode of the case study

Input: A GIF file containing multiple frames
Output: GIF files with outlined objects, skeletons of objects, prominent objects circled, and motion detection

```
1. Function process_gif(input_gif):
2.     frames = load_gif(input_gif)
3.     max_width, max_height = get_max_dimensions(frames)
4.     outlined_frames = []
5.     skeletons_frames = []
6.     prominent_objects_frames = []
7.     motion_detection_frames = []
8.     previous_skeleton = None
9.     for frame in frames:
10.        resized_frame = resize_frame(frame, max_width, max_height)
11.        outlined_frame = create_outline(resized_frame)//Outline Creation
12.        outlined_frames.append(outlined_frame)
13.        skeleton = create_skeleton(outlined_frame)//Skeleton Creation
14.        skeletons_frames.append(skeleton)
15.        prominent_object_frame = highlight_prominent_object(outlined_frame)//Prominent
16.        prominent_objects_frames.append(prominent_object_frame)//Object Highlighting
17.        if previous_skeleton is not None://Motion Detection
18.            motion_frame = detect_motion(previous_skeleton, skeleton)
19.            motion_detection_frames.append(motion_frame)
20.        previous_skeleton = skeleton
21.    outlined_objects_gif = save_gif(outlined_frames)
22.    skeletons_gif = save_gif(skeletons_frames)
23.    prominent_objects_gif = save_gif(prominent_objects_frames)
24.    motion_detection_gif = save_gif(motion_detection_frames)
25.    return outlined_objects_gif, skeletons_gif, prominent_objects_gif, motion_detection_gif
```

In this pseudocode:

load_gif() is responsible for loading the GIF and converting its frames to a format suitable for processing, *get_max_dimensions()* calculates the maximum width and height across all frames to ensure consistent sizing, *resize_frame()* resizes a frame to the maximum dimensions, *create_outline()* generates an outline of objects in a frame, *create_skeleton()* computes the skeleton of objects in a frame. *highlight_prominent_object()* highlights the most prominent object in a frame, *detect_motion()* detects motion between two consecutive frames based on their skeletons, *save_gif()* saves a list of frames as a GIF file.

Figure 9 demonstrates the Python modules installed and imported based on the suggestion from ChatGPT.

As can be seen from Figure 9, the most used libraries are mentioned in it. There are no rare or private modules and/or tools.

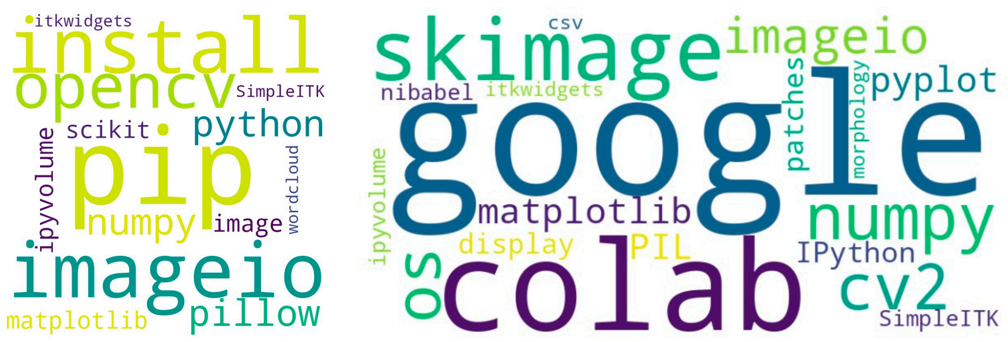


Figure 9. Sentiment analysis of the libraries installed (**left**) vs. libraries imported (**right**).

Based on the recommendations provided by the bot, a comprehensive approach was adopted to enhance the object outlining process. This approach includes four distinct methods, each contributing to a more refined and smoother object outline: *Gaussian Blur* [25], a technique renowned for its efficacy in image smoothing; *Canny Edge Detection* [26], an advanced edge-detection method that outperforms basic thresholding; *Morphological Operations* [27], a sequence of dilation followed by erosion that aids in bridging small gaps and sealing minor holes in the contours, contributing to a more cohesive and uninterrupted outline; and *Contour Approximation* [28], a method that works by compressing horizontal, vertical, and diagonal segments, retaining only their endpoint.

Figure 10 illustrates the transformative impact of these methods. The left side of the figure presents the original image of a running cheetah [29], while the right side showcases the refined outline of the cheetah, achieved through the application of the methods mentioned above.

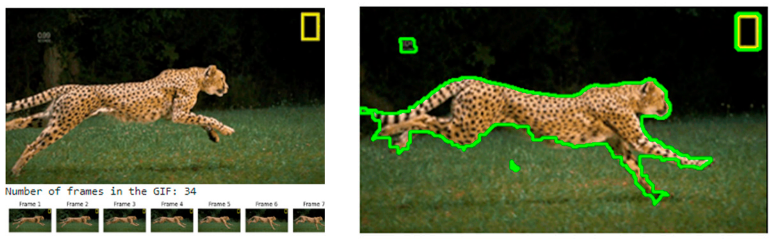


Figure 10. Split the original GIF animation into frames (**left**) and create its outline (**right**).

Adding Contour Approximation significantly changes the outcome, making the result less smooth, as seen in Figure 11 (left). At the same time, the amount of noise seems to be reduced compared to the result of the three other methods combined.

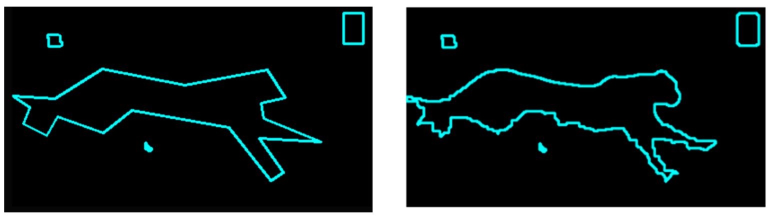


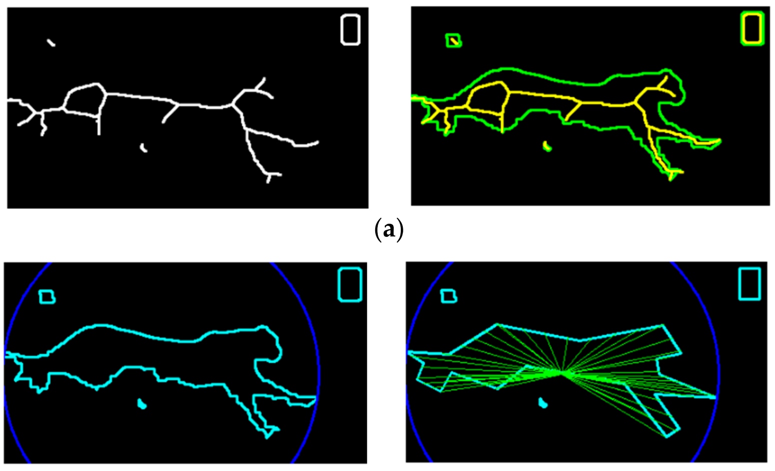
Figure 11. Outline with (**left**) vs. without (**right**) Contour Approximation.

The pseudocode of Outline creation without Contour Approximation can be observed in Algorithm 2.

Algorithm 2: Object Outlining in GIF Frames

```
Input: GIF file
Output: GIF with outlined objects
frames ← [frame for frame in imageio.get_reader(GIF path)]//Read GIF Frames
max_width ← max([frame.width for frame in frames])//Determine Frame Dimensions
max_height ← max([frame.height for frame in frames])//to standardize the frame size
outlines ← []//Initialize Output: Create a list to store the outlined frames.
for frame in frames do//Process Each Frame
    frame ← resize_with_padding(frame, max_width, max_height)//Resize Frame
    gray ← cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)//Convert to Grayscale
    blurred ← cv2.GaussianBlur(gray, (5, 5), 0)//Apply Gaussian Blur
    edges ← cv2.Canny(blurred, 50, 150)//the Canny edge detector.
    kernel ← np.ones((5, 5), np.uint8)//Morphological Operations
    dilated ← cv2.dilate(edges, kernel, iterations = 2)//Apply dilation
    eroded ← cv2.erode(dilated, kernel, iterations = 2)//Apply erosion to close gaps
    //Find Contours: Extract contours from the processed image.
    contours, _← cv2.findContours(eroded, cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
    outline ← cv2.drawContours(frame.copy(), contours, −1, (0, 255, 0), 2)//Draw the contours
    outlines.append(outline)//Add the outlined frame to the list
end for
Save as GIF: Save the outlined frames as a new GIF file.
```

To further reduce the noise, the researchers applied their approach 'Cheetah in the circle'. The most prominent object, the one with the most significant area, was put in the circle, and everything that appeared outside the circle was not considered to be a part of the motion. Figure 12a,b highlighted the results.



(**a**)

(**b**)

Figure 12. (**a**) Skeleton without (**left**) vs. with object outline (**right**). (**b**) Identifying the prominent object surrounded by a circle (**left**) and reduced noise revision (**right**).

To further specify the method, with the current name *Cheetah in the circle*, the pseudocode of the method is provided below. As can be observed from 16.2, the center of the circle surrounding the object also plays a role in the method, the lines coming out from the center of the circle to the outline of the object.

To validate the results, other images were tested against the same script.

Upon obtaining the skeletonized output from the Skeleton app, as demonstrated in Algorithm 3, various avenues exist for further processing and applications. Potential uses and subsequent steps once the skeleton is obtained include:

- *Motion Analysis and Tracking*: the skeleton output can be used to analyze or track the movement of the prominent object in each frame. This analysis is essential in biomechanics or animation fields, where understanding movement dynamics is crucial [30].
- *Machine Learning and Feature Extraction*: the obtained skeletons can serve as a simplified yet effective feature set for machine learning algorithms, particularly in image recognition or classification tasks [31].
- *Object Recognition and Categorization*: the structural information provided by the skeleton can be instrumental in recognizing or categorizing different objects within various frames [32].
- *3D Modeling and Visualization*: skeleton data can inform 3D modeling processes, aiding in the construction of detailed visualizations or reconstructions, especially in fields requiring a deep understanding of an object's structure [33].
- *Development of Control Systems in Robotics*: skeletal data can be pivotal in developing algorithms for robotic movements or gesture-based control systems, enhancing the interaction between robots and their environment [34].
- *Augmented Reality (AR) Applications*: leveraging the skeleton data can provide a unique interactive experience, overlaying digital information onto real-world objects [35].

Algorithm 3: Cheetah in the circle

```
Input: The list of frames
Output: A new GIF with circles drawn around the prominent object in each frame.
For each frame, process to highlight the prominent object:
    Resize the frame with padding to match the maximum dimensions.
    Convert the frame to grayscale.
    Apply Gaussian Blur to reduce noise.
    Apply thresholding to create a binary image.
    Find contours in the binary image.
    Filter out small contours based on a size threshold to remove noise.
    Identify the largest contour, assuming it is the prominent object.
    Draw a circle around the largest contour.
    Save the frame.
Save the processed frames with circles as a new GIF.
```

6. Discussion

While summarizing the study results, the researchers would like to present a comparative analysis of semi-automated and fully automated approaches in the context of code conversion from MATLAB to Python to justify the approach taken. This analysis is structured around criteria such as efficiency, accuracy, ease of implementation, flexibility, and user involvement, as shown in Table 4. Fully automated approaches in code conversion or similar computational tasks typically involve advanced algorithms or AI models that can perform tasks with minimal or no human intervention. Examples include deep learning models, where AI models were trained on large datasets to understand and translate code from one programming language to another without human aid, and rule-based conversion tools, where some software used a set of predefined rules to automatically convert code and neural machine translation for code. These AI-driven tools can translate programming languages by learning from vast examples of code conversions [36]. Presented below, Table 4 aids in understanding the trade-offs between the two approaches.

Table 4. Comparative Analysis of Semi-Automated vs. Fully Automated Methods.

Criteria	Semi-Automated Approach	Fully Automated Approach
Efficiency	Time-intensive but allows for greater control during the conversion process.	Faster, as it requires less human intervention.
Accuracy	High, due to human oversight correcting potential errors.	May vary, depending on the sophistication of the automation algorithms.
Ease of Implementation	More complex, requiring a balance between automated tools and human input.	Simpler, as it relies heavily on pre-defined algorithms.
Flexibility	Highly adaptable to specific requirements of different projects.	Less adaptable, often designed for general cases.
User Involvement	Requires significant user involvement for decision-making and error correction.	Minimal user input is needed, as the process is largely self-sufficient.

As highlighted in the table, fully automated methods offer faster processing and require less human input; at the same time, they may lack precision and adaptability. At the current level of the Large Language Models (LLMs) development, the accuracy they provide and the way they work through the semi-automated approach, with its combination of automated tools and human oversight, chosen by the authors, ensures higher accuracy through manual checks and corrections. It also offers greater flexibility to tailor the process for specific project needs. This balance makes the semi-automated approach more suitable for complex and nuanced tasks such as the conversion from MATLAB to Python, a sophisticated engineering skeletonization code, and the topic of this study.

The researchers also want to emphasize the importance of AI pair programming used in this study. While it became a very common and relevant practice in recent years [37], its primary purpose was to leverage the strengths of artificial intelligence (AI) alongside human expertise. In this method, an AI, such as ChatGPT, assists a human programmer, combining the AI's vast data-processing capabilities with the programmer's contextual understanding and problem-solving skills. This synergistic approach enhances the code conversion process's efficiency, accuracy, and innovation. AI can suggest code, identify errors, and offer solutions based on its extensive training, while the human programmer provides the critical thinking and contextual decisions necessary for complex tasks such as MATLAB to Python code conversion. While discussing the role of the ChatGPT bot/model in this study, it is crucial to recognize the unique capabilities of this unique AI tool. ChatGPT was not a helper but a true collaborative partner. It played a crucial role in generating Python code from MATLAB scripts, meticulously ensuring syntax accuracy, and preserving the original algorithmic intent. Its integration with other code conversion tools was seamless, acting as an intelligent assistant that provided real-time feedback and suggestions, thereby enhancing the overall efficiency of the conversion process. By offering innovative solutions and alternative approaches, ChatGPT helped navigate through coding challenges that often required a blend of computational efficiency and human ingenuity. The impact of ChatGPT on the project was evident in the tangible improvements in both the speed and accuracy of the code conversion, leading to a more streamlined and error-resistant output than traditional methods. However, with their extensive experience, the researchers also acknowledged the limitations of ChatGPT. They noted instances where human intervention was essential to resolve complex scenarios beyond the scope of the AI's current capabilities. This balanced perspective highlights the team's understanding of the symbiotic relationship between AI tools and human expertise, particularly in MATLAB to Python code conversion.

7. Conclusions

This section introduces a structured point-by-point summary of the essential findings and implications of the research. It addresses each of the stated research questions (RQ1, RQ2, RQ3, RQ4), emphasizing the significant role of LLMs in the code conversion process and app development. Concluding remarks:

- LLMs as AI Pair Programmers (RQ1): This study confirms that Large Language Models, such as ChatGPT-4, Bard, and Bing, can be effectively used as AI pair programmers in MATLAB-to-Python conversion for complex projects. These models excel in understanding the logic and structure of the code, translating MATLAB code into Python with high accuracy, and providing valuable insights into code functionality.
- Comparison with Existing Tool (RQ2): The research shows that LLMs outperform existing language-to-language converter tools regarding accuracy and efficiency. In handling complex MATLAB code, LLMs produced functional, accurate, and efficient Python code more effectively and swiftly than current tools.
- Automation Using OpenAI API (RQ3): The study explored the automation of the MATLAB-to-Python conversion process using the OpenAI API. It identified challenges such as token limitations for large code segments and occasional inaccuracies in complex translations. The research suggests segmenting code for translation and combining the API's output with manual review as effective solutions.
- Feasibility of Skeleton App Development (RQ4): The feasibility of creating a functional Skeleton App with the help of ChatGPT post-M-to-PY translation is confirmed. The Skeleton App, available online at no cost, is user-friendly and adept at generating image skeletons, showcasing the potential of LLMs in code translation and application development.
- Commercial Considerations: While using ChatGPT-4 and similar models incur costs, this was not a scalability concern for this project. However, it may become relevant for larger applications.
- Shift Towards Testing Efforts: As the reliability of code generation by LLMs increases, this research highlights a shift towards enhancing testing efforts, where LLMs can provide significant assistance.
- Overall Impact and Future Applications: This research validates the latest LLMs in-app generation advanced capabilities for image sketch abstractions and 3D motion detections. The developed M-to-PY converter bridges the gap between MATLAB scientific code and AI code in Python, facilitating the scientific use of AI technologies.

Author Contributions: Conceptualization, Y.K. and Z.G.; methodology, Y.K. and Z.G.; software, Z.G.; validation, Z.G., O.A. and J.L.; formal analysis, J.L. and K.L.; investigation, O.A.; resources, K.L.; data curation, O.A.; writing—original draft preparation, Y.K. and Z.G.; writing—review and editing, J.L., K.L. and L.N.; visualization, Y.K., Z.G. and O.A.; supervision, L.N. and P.M.; project administration, P.M.; funding acquisition, Y.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by NSF awards 1834620 and 2137791 and Kean University.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. OpenAI Quickstart. 2023. Available online: https://platform.openai.com/docs/quickstart (accessed on 1 September 2023).
2. Leonard, K.; Morin, G.; Hahmann, S.; Carlier, A. A 2D shape structure for decomposition and part similarity. In Proceedings of the 2016 23rd International Conference on Pattern Recognition (ICPR), Cancun, Mexico, 4–8 December 2016; pp. 3216–3221. [CrossRef]
3. Javaid, M.A. Understanding Dijkstra's Algorithm. *SSRN Electron. J.* **2013**, 14–26. [CrossRef]
4. Dennis Layton. Open AI Code Interpreter. 2023. Available online: https://medium.com/@dlaytonj2/open-ai-code-interpreter-what-you-need-to-know-29c57085835e (accessed on 1 September 2023).
5. OMPC. 2008. Available online: http://ompc.juricap.com/ (accessed on 1 September 2023).
6. Meta, A.I. Deep Learning to Translate between Programming Languages. 2020. Available online: https://ai.facebook.com/blog/deep-learning-to-translate-between-programming-languages/ (accessed on 1 September 2023).
7. Koziolek, H.; Gruener, S.; Ashiwal, V. ChatGPT for PLC/DCS Control Logic Generation. *arXiv* **2023**, arXiv:2305.15809.
8. Li, R.; Pu, C.; Fan, F.; Tao, J.; Xiang, Y. Leveraging ChatGPT for Power System Programming Tasks. *arXiv* **2023**, arXiv:2305.11202.
9. Tsai, M.L.; Ong, C.W.; Chen, C.L. Exploring the use of large language models (LLMs) in chemical engineering education: Building core course problem models with Chat-GPT. *Educ. Chem. Eng.* **2023**, *44*, 71–95. [CrossRef]
10. Merow, C.; Serra-Diaz, J.M.; Enquist, B.J.; Wilson, A.M. AI chatbots can boost scientific coding. *Nat. Ecol. Evol.* **2023**, *7*, 960–962. [CrossRef] [PubMed]
11. Kumar, Y.; Morreale, P.; Sorial, P.; Delgado, J.; Li, J.J.; Martins, P. A Testing Framework for AI Linguistic Systems (testFAILS). *Electronics* **2023**, *12*, 3095. [CrossRef]
12. Maurer, B.; Geo, P. Tech, AI Chatbot Geotechnical Engineer: How AI Language Models Like "ChatGPT" Could Change the Profession. *Engrxiv* **2023**. [CrossRef]
13. Crokidakis, N.; de Menezes, M.A.; Cajueiro, D.O. Questions of science: Chatting with ChatGPT about complex systems. *arXiv* **2023**, arXiv:2303.16870.
14. Nikolic, S.; Daniel, S.; Haque, R.; Belkina, M.; Hassan, G.M.; Grundy, S.; Sandison, C. ChatGPT versus engineering education assessment: A multidisciplinary and multi-institutional benchmarking and analysis of this generative artificial intelligence tool to investigate assessment integrity. *Eur. J. Eng. Educ.* **2023**, *48*, 559–614. [CrossRef]
15. Vercel. AI Code Translator. 2023. Available online: https://vercel.com/templates/next.js/ai-code-translator (accessed on 1 September 2023).
16. CodeConvert. 2023. Available online: https://www.codeconvert.ai/ (accessed on 1 September 2023).
17. Ma, J.; Ren, X.; Li, H.; Li, W.; Tsviatkou, V.Y.; Boriskevich, A.A. Noise-against skeleton extraction framework and application on hand gesture recognition. *IEEE Access* **2023**, *11*, 9547–9559. [CrossRef]
18. Wang, T.; Yamakawa, Y. Edge-Supervised Linear Object Skeletonization for High-Speed Camera. *Sensors* **2023**, *23*, 5721. [CrossRef]
19. Lovanshi, M.; Tiwari, V. Human skeleton pose and spatio-temporal feature-based activity recognition using ST-GCN. *Multimed. Tools Appl.* **2023**. [CrossRef]
20. Usmani, A.; Siddiqui, N.; Islam, S. Skeleton joint trajectories based human activity recognition using deep RNN. *Multimed. Tools Appl.* **2023**. [CrossRef]
21. Skeleton App. 2023. Available online: https://skeleton-app-65563db1db74.herokuapp.com/ (accessed on 1 September 2023).
22. Skeleton App Repo. 2023. Available online: https://github.com/ykumar2020/SkeletonApp (accessed on 1 November 2023).
23. Discussion with ChatGPT-4 on Motion Detection. Available online: https://chat.openai.com/share/9f06d172-d095-4199-95d3-40406ea00ed7 (accessed on 1 September 2023).
24. Python Code and Its Results. 2023. Available online: https://colab.research.google.com/drive/1yy3h4_FGTsDOf5wOTFs34p5GXMn_QU1X?usp=sharing (accessed on 1 September 2023).
25. Bergstrom, A.C.; Conran, D.; Messinger, D.W. Gaussian blur and relative edge response. *arXiv* **2023**, arXiv:2301.00856.
26. Canny, J. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **1986**, *6*, 679–698. [CrossRef]
27. Soille, P. *Morphological Image Analysis: Principles and Applications*; Springer: Berlin, Germany, 1999; Volume 2, pp. 170–171.
28. Chakraborty, D. OpenCV Contour Approximation (cv2.approxPolyDP). PyImageSearch. 2021. Available online: https://pyimagesearch.com/2021/10/06/opencv-contour-approximation/ (accessed on 15 January 2024).
29. GIF Image of a Running Cheetah. Available online: https://i.pinimg.com/originals/61/5d/b5/615db596fe3db4b7e04b54af1aea5826.gif (accessed on 15 January 2024).
30. Zarka, N.; Alhalah, Z.; Deeb, R. Real-Time Human Motion Detection and Tracking. In Proceedings of the 2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications, Damascus, Syria, 7–11 April 2008; pp. 1–6. [CrossRef]
31. Ali, M.; Kumar, D. A Combination between Deep learning for feature extraction and Machine Learning for Recognition. In Proceedings of the 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kharagpur, India, 6–8 July 2021; pp. 01–05. [CrossRef]
32. Carvalho, L.E.; von Wangenheim, A. 3D object recognition and classification: A systematic literature review. *Pattern Anal. Appl.* **2019**, *22*, 1243–1292. [CrossRef]
33. Cárdenas, J.L.; Ogayar, C.J.; Feito, F.R.; Jurado, J.M. Modeling of the 3D Tree Skeleton Using Real-World Data: A Survey. *IEEE Trans. Vis. Comput. Graph.* **2023**, *29*, 4920–4935. [CrossRef] [PubMed]

34. Yang, N.; Duan, F.; Wei, Y.; Liu, C.; Tan, J.T.C.; Xu, B.; Zhang, J. A study of the human-robot synchronous control system based on skeletal tracking technology. In Proceedings of the 2013 IEEE International Conference on Robotics and Biomimetics (ROBIO), Shenzhen, China, 12–14 December 2013; pp. 2191–2196. [CrossRef]
35. Singh, J.; Urvashi; Singh, G.; Maheshwari, S. Augmented Reality Technology: Current Applications, Challenges and its Future. In Proceedings of the 2022 4th International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India, 21–23 September 2022; pp. 1722–1726. [CrossRef]
36. Yang, X.; Wang, X.; Liu, Z.; Shu, F. M2Coder: A Fully Automated Translator from Matlab M-functions to C/C++ Codes for ACS Motion Controllers. In Proceedings of the International Conference on Guidance, Navigation and Control, Tianjin, China, 5–7 August 2022; Springer Nature: Singapore, 2022; pp. 3130–3139.
37. Benefits of AI Tools Section at Stack Overflow Developer Survey. 2023. Available online: https://survey.stackoverflow.co/2023/#benefits-of-ai-tools (accessed on 15 January 2024).