ChatGPT Translation of Program Code for Image Sketch Abstraction

Yulia Kumar *, Zachary Gordon *, Oluwatunmise Alabi *, J. Jenny Li *, Kathryn Leonard, Linda Ness, Patricia Morreale *

Posted Date: 30 October 2023

doi: 10.20944/preprints202310.1906.v1

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

Article

ChatGPT Translation of Program Code for Image Sketch Abstraction

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

[1] Kean University (Union, NJ, USA); ykumar@kean.edu, gordonza@kean.edu, oalabi@kean.edu, juli@kean.edu
[2] Occidental College (Los Angeles, CA, USA); leonardk@oxy.edu
[3] Rutgers University (New Brunswick, NJ, USA); ness.linda@gmail.com
* Correspondence: ykumar@kean.edu; Tel.: (908) 3136015)

Abstract: The migration from MATLAB to Python (M-to-PY) has gained significant traction in recent computational research. While MATLAB has long served as a linchpin in myriad scientific endeavors, there's an emerging trend to rejuvenate these projects using Python's extensive AI tools and libraries. This study presents a semi-automated process for M-to-PY conversion, using a detailed case study of an image skeletonization project comprising fifteen MATLAB files and a 1404-image dataset. Skeletonization is foundational for ongoing 3D motion detection research using AI transformers, predominantly developed in Python. The utilization of ChatGPT-4, acting as an AI co-programmer, is pivotal in this conversion. By leveraging the public OpenAI API, we developed an M-to-PY converter prototype, evaluated its efficacy using test cases from the Bard bot, and subsequently employed the converted code in an AI application. The dual contributions encompass a well-tested M-to-PY converter and a Skeleton App capable of sketching and skeletonizing any given image, enriching the AI toolset. This study accentuates how AI resources, like ChatGPT-4, can simplify code transitions, opening doors for innovative AI implementations using primarily MATLAB-coded scientific research.

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

---

The innovative Skeleton app accepts an image of any object and draws its sketch and its skeleton. It also consists of the innovative MATLAB to Python converter.

1. Introduction

Transitioning from MATLAB to Python, 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?

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 October 2023 doi:10.20944/preprints202310.1906.v1

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?

The 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:

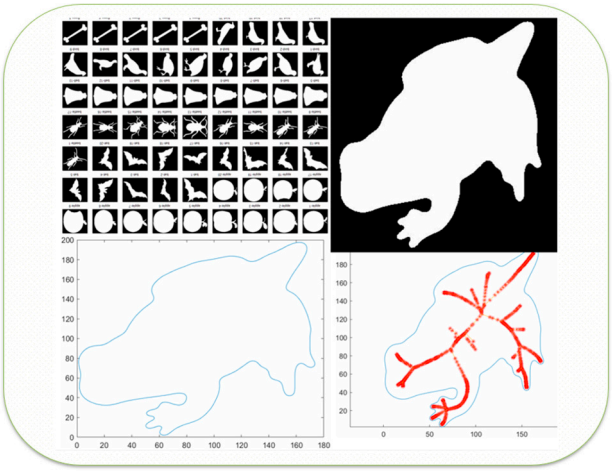

Figure 1. Blum skeletonization produced by MATLAB: a snippet of the contour dataset (top left), frog's contour (top right), sketch, and skeleton (bottom left and right).

Figure 1 depicted only a small part of the initial dataset. It consists of 1404 contour images such as the one of a frog. It required time and code upgrade efforts to make the original MATLAB code work, due to an inconsistency of software tools and code versions. As the initial object-oriented MATLAB code had complex workarounds the researchers took that are specifically optimized for use in MATLAB data structures, such as using complex numbers to represent two-dimensional arrays, which caused additional challenges to the team of junior developers debugging the code before using the release of LLMs like ChatGPT. The original MATLAB code used several MATLAB-specific functions like "*inpolygon*()" and had a custom Dijkstra algorithm, as explained by Adeel Javaid [3], implementation. The core of the code is to find boundary coordinates and to calculate a set of medial axis points for each input image, called image sketch abstraction or object skeletonization. Once this is accomplished the skeleton is drawn over each boundary image. This skeleton designates the object's center of gravity and can be used to track motion in AI research, what can be seen later in the paper. The conversion of the project to Python can benefit the area of 3D motion detection with the help of transformers and might be of great interest to other researchers.

The case study used mainly ChatGPT-4 as an AI Pair Programmer tool throughout all conversion processes with minor use of Bard and Bing chatbots. It involved chatting with the bot on 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 for it. OpenAI API and ChatGPT-4 model are then used for live M-to-PY conversion, available through the Skeleton App. The app logo was generated by the DALL-E Open AI tool, 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 the realm of 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

 doi:10.20944/preprints202310.1906.v1

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 like 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 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 like ChatGPT on various professions. Brett Maurer [12] discussed how AI chatbots like 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 benchmarking and analysis of ChatGPT to investigate assessment integrity in engineering education. These studies found that ChatGPT was able to generate code as accurately as code created by software developers, making it hard to distinguish between the two.

While developing our 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 ChatGPT-4 model for code translation. However, our work is distinct as we 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 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

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 October 2023 doi:10.20944/preprints202310.1906.v1

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

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

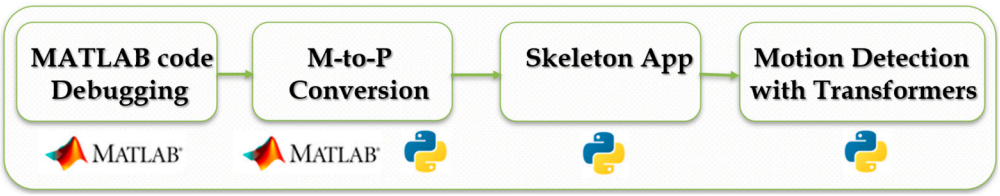

Figure 2. Project plan with M-to-PY translation and the Skeleton App as middle-tiers of the study.

Figure 2 shows the project plan consisting of four main steps, with additional implementation in between. 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 powerful Windows Alienware machine to which several research team members could connect remotely. Before using 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.

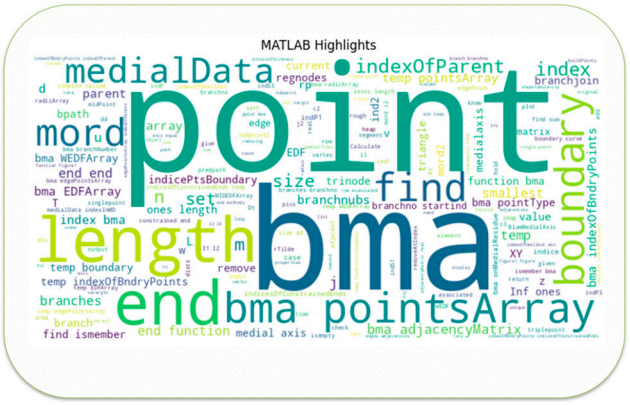

Figure 3. Sentiment analysis of the MATLAB code.

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 center of gravity of the target object, followed by the creation of 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 important 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

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 October 2023 doi:10.20944/preprints202310.1906.v1

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 a Skeleton app eventually.

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, M-to-PY converter is residing on the M-to-PY page of the app. 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 we are undecided about long-term data storage. Figure 4 demonstrates the Home and Result web pages, and Figure 5 demonstrates the M-to-PY translation page of the app.

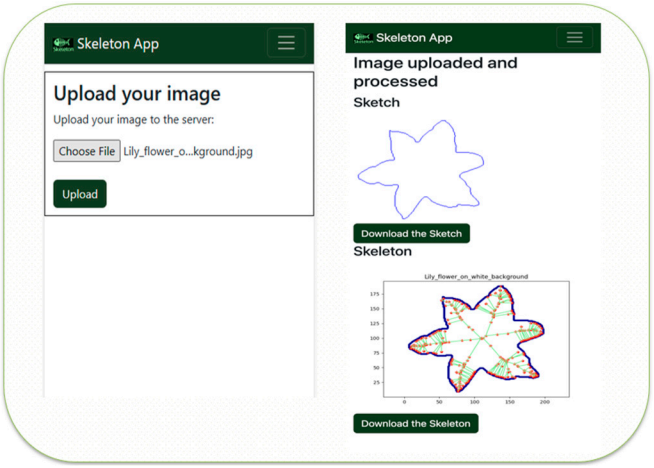

Figure 4. Home and Result web pages of the Skeleton app.

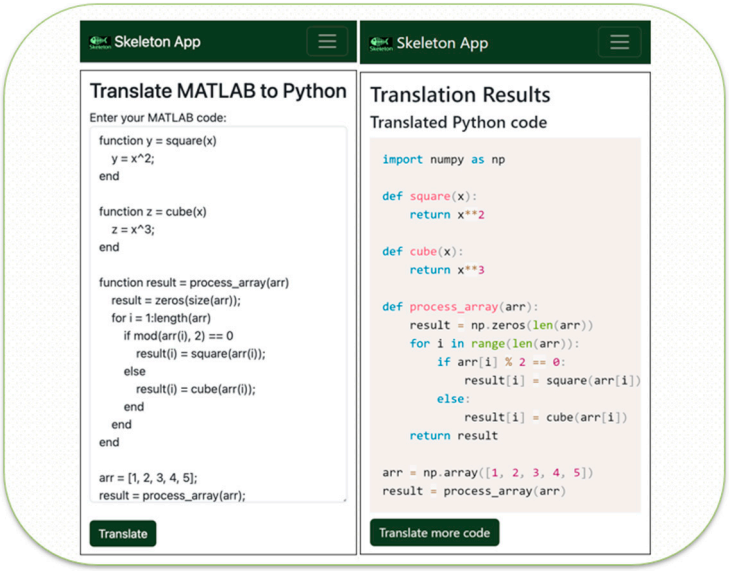

Figure 5. M-to-PY translation page of the app.

The translation, presented in Figure 5 is done by the OpenAI API and ChatGPT-4 model. As can be seen from Figures 4 and 5, 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.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 October 2023 doi:10.20944/preprints202310.1906.v1

The Skeleton app architecture can be observed in Figure 6. It is novice and unique, it can scale very fast if deployed live.

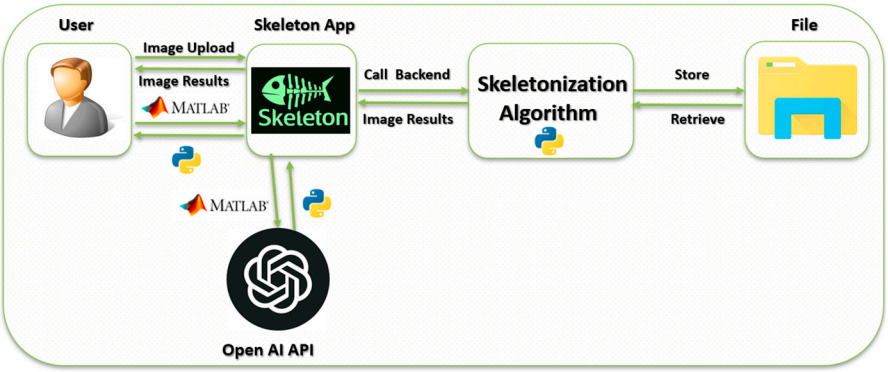

Figure 6. The Skeleton app architecture.

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

Currently, the last stage of the project is in progress, we are planning to publish it as a separate study unrelated to the M-to-PY conversion and the Skeleton app. There is recently substantial progress in motion detection, and it is important to understand the most relevant approaches and how our approach will be different 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 7 shows one of the testing outcomes. By pinpointing a subject's center of gravity and creating a skeleton and outline for the object, we eventually anticipate significant enhancements in motion tracking accuracy compared to existing methodologies.

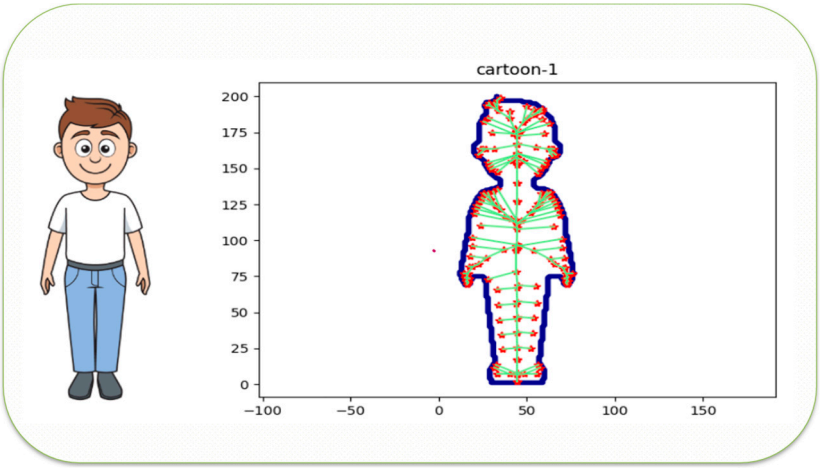

Figure 7. The cartoon person and the generated skeleton.

4. Methodology of M-to-PY and the App

4.1. The M-to-PY methodology

The methodology for the M-to-PY conversion for a complex project like a One-step Skeletonization is an iterative process, that can be reproduced step-by step. The nine key steps are presented in Figure 8. The selection of LLMs is a critical step in our methodology. We 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

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 October 2023 doi:10.20944/preprints202310.1906.v1

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.

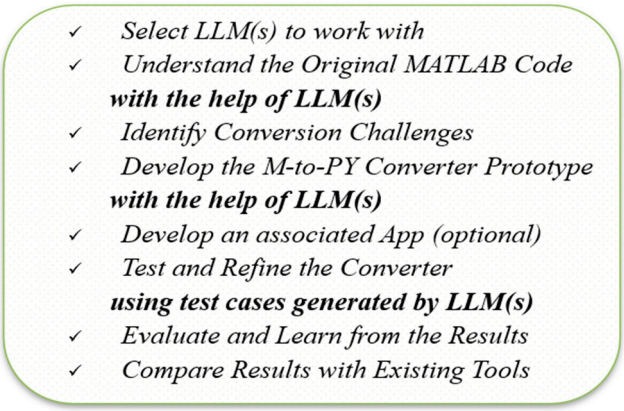

Figure 8. The M-to-PY methodology steps.

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 we 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 speeded 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 Skeleton app logo was developed by DALL-E 2 text-to-image generation tool. Key points to consider are simplicity, accessibility, responsiveness, equitability, and inclusivity as described by Pankati Patel, et. al. [22,23].

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, all three ChatGPT-4, Bard, and Bing were asked to generate high-view-point test cases, then Bard was chosen as the best creator of those, and then ChatGPT-4 produced the actual code to test. Figure 9 represents test cases at-a-Glance, complete code can be found online [24].

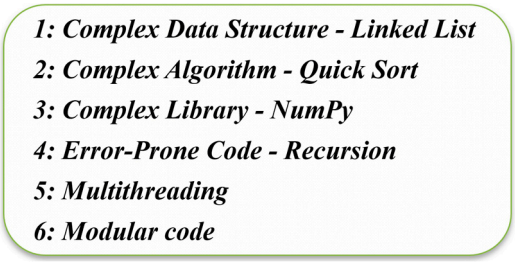

Figure 9. Test cases proposed by Bard.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 October 2023 doi:10.20944/preprints202310.1906.v1

While it seems that Bing was an outsider in this process, its ability to browse the internet was very important in the study. Obtaining the latest AI and development news and tools is critical to be compatible and relevant in the research study. The Skeleton app in its ability to create sketches and the skeleton was straightforward. 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 latest AI algorithms most often written in Python. The 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 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 our 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.

Converter Tool	Test case number					
	1	2	3	4	5	6
M-to-PY	✓	✓	✓	✓	✓	✓
Vercel	✓	✓	✓	✓	✓	✗
CodeConvert AI	✓	✓	✓	✓	✗	✗
OMPC	✗	✗	✗	✗	✗	✗

The methodology for the development of the Skeleton App, which is a part of the M-to-PY conversion project, is also an iterative process. It is in fact a by-product of the converter that was used to test the effectiveness, accuracy, and performance of the generated converter. Figure 10 shows the workflow of the generation of the Skeleton App, including eight key steps. The one prototype development step was generated by an LLM model, and the rest steps were conducted manually with the assistance of various LLM models as described earlier.

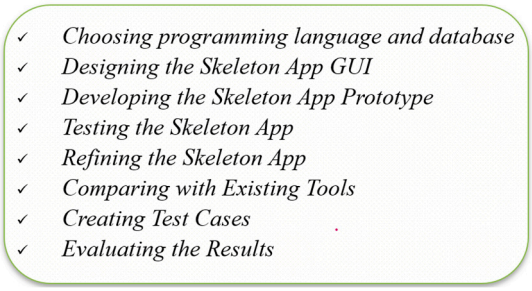

Figure 10. M-to-PY methodology steps.

As OpenAI API provides QuickStart tutorials only for React.js JavaScript and Python Flask web apps, we chose the second one to be consistent with the M-to-PY converter. Since the converter itself was AI generated, it would be straightforward to create a JavaScript version if needed. As shown in Figure 10, the new converter will need to go through thorough testing and refinement once generated. With this paradigm of AI code generation, software engineering and software development will ship the focus from coding to testing and maintenance.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 October 2023 doi:10.20944/preprints202310.1906.v1

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, generation of test cases by LLMs, and the capacity of LLMs for programming language-to-language translation. After the integration of 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 there are few intelligent models like ChatGPT-4 that 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 we 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. [25]. 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. 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 LLMs 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 at once due to ChatGPT-3 limitations on both input and output. As the code is object-oriented and consists 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 an additional level of complexity. The code was translated in small chunks and most of the debugging and troubleshooting had to be done by a human copilot. 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 for human debugging was used with little success. A significant problem was situations when the shape or dimensions of data structures returned from a function in Python (aka arrays) did not match the MATLAB code. There were many cases of array vs list data structures encountered that 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.

Main test cases of our experiments are in Table 2.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 October 2023 doi:10.20944/preprints202310.1906.v1

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 generates not only different variable names but assumes and produces 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 here 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 rather 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 increases.

The release of ChatGPT-4 was the event that made the study successful. The level of the model "understanding" 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 their 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 going back after several hours or to use a 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 was able not only to explain the code, explain the errors but 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 11, it is the ChatGPT-4 model that produces the complete code for a function. The results of the successful run can be observed in Figure 12:

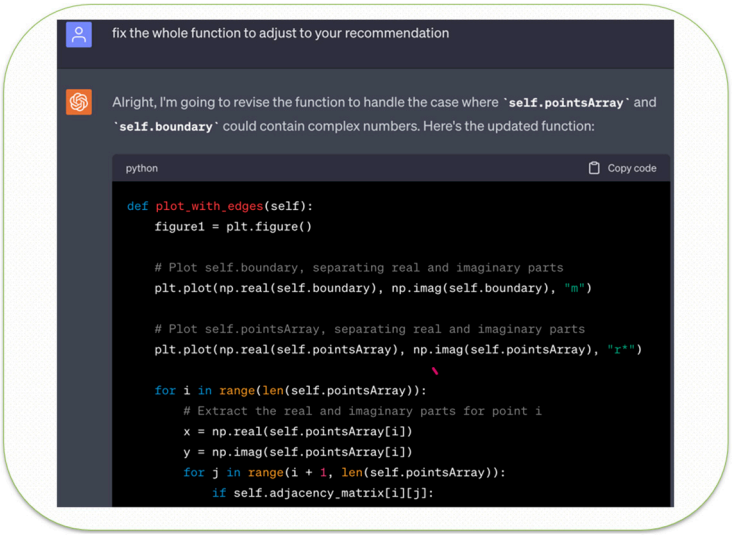

Figure 11. Code revision by ChatGPT.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 October 2023 doi:10.20944/preprints202310.1906.v1

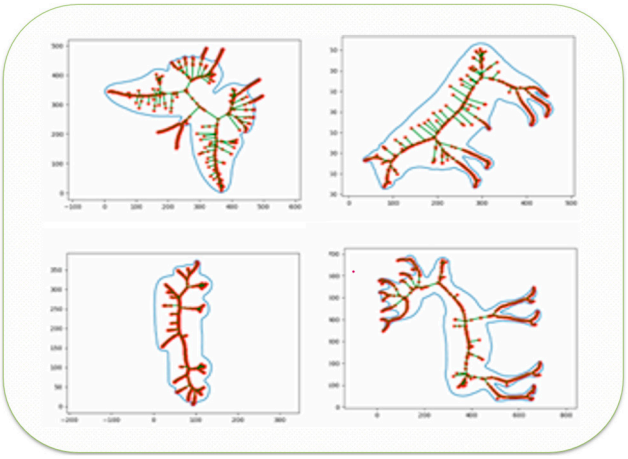

Figure 12. Skeleton images produced by the Python code.

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. Interesting that Bard bot happily generated the test scenarios that were evaluated as more solid than ChatGPT's ones, but the model refused to provide the actual code snippets to test. The study's finding is that this step was successfully accomplished by combining 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 the need for 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 can 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 the code's functionality and add comments to it. This was particularly helpful in understanding complex parts of the code. The LLM was also able to identify and correct 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 task that had previously been challenging 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. We 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, we observed improved quality of the translated code with each new version of ChatGPT.

We 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 frontend, with the assistance of ChatGPT-4's Code Interpreter and DALL-E text-to-image generator. The app demonstrates the practical application of the M-to-PY conversion process.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 October 2023 doi:10.20944/preprints202310.1906.v1

Throughout the project, we kept track of our progress and documented our findings. We compared the performance of our M-to-PY converter with existing tools and created complex test cases to evaluate its performance.

We 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 takes 4.32 milliseconds.

The research provides valuable insights into the capabilities and limitations of LLMs. It demonstrates their potential in facilitating the development of practical apps. Our 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 M-to-PY converter together with the Skeleton app were fully tested and deployed [21] 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 set to do an Advanced Data Analysis, last updated on September 25, 2023, was asked to walk 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 [26,27]. More examples can be provided on demand.

Table 2 explains the applications, pros, and cons of creating Object Outlines and Skeletons in such a study.

Table 2. Object Outlines vs Skeletons for Motion Detection.

Criteria	Object Outlines	Skeletons
Representation	Complete boundary of the object	Simplified, centerline representation of the object
Pros	Clear understanding of shape and orientation Easy calculation of features like area and perimeter	Simplified analysis of complex shapes Less sensitive to boundary noise Useful for analyzing 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 movement of specific object parts

As can be seen from Table 2. 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 guidance of ChatGPT can be seen below:

Algorithm 1: High-view pseudocode of the case study

Input: A GIF file containing multiple frames

Output: GIFs files with outlined objects, skeletons of objects, prominent objects circled, motion detection

```
1. Function process_gif(input_gif):
2.     frames = load_gif(input_gif)
3.     max_width, max_height = get_max_dimensions(frames)
4.
5.     outlined_frames = []
6.     skeletons_frames = []
7.     prominent_objects_frames = []
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 October 2023 doi:10.20944/preprints202310.1906.v1


```
motion_detection_frames = []

previous_skeleton = None
for frame in frames:
    resized_frame = resize_frame(frame, max_width, max_height)

    # Outline Creation
    outlined_frame = create_outline(resized_frame)
    outlined_frames.append(outlined_frame)

    # Skeleton Creation
    skeleton = create_skeleton(outlined_frame)
    skeletons_frames.append(skeleton)

    # Prominent Object Highlighting
    prominent_object_frame = highlight_prominent_object(outlined_frame)
    prominent_objects_frames.append(prominent_object_frame)

    # Motion Detection
    if previous_skeleton is not None:
        motion_frame = detect_motion(previous_skeleton, skeleton)
        motion_detection_frames.append(motion_frame)

    previous_skeleton = skeleton

outlined_objects_gif = save_gif(outlined_frames)
skeletons_gif = save_gif(skeletons_frames)
prominent_objects_gif = save_gif(prominent_objects_frames)
motion_detection_gif = save_gif(motion_detection_frames)

return outlined_objects_gif, skeletons_gif, prominent_objects_gif, motion_de
    tection_gif
prominent_objects_gif = save_gif(prominent_objects_frames)
motion_detection_gif = save_gif(motion_detection_frames)

return outlined_objects_gif, skeletons_gif, prominent_objects_gif, motion_de
tection_gif
```

In this pseudocode:

load_gif function is responsible for loading the GIF and converting its frames to a format suitable for processing.

get_max_dimensions function calculates the maximum width and height across all frames to ensure consistent sizing.

resize_frame function resizes a frame to the maximum dimensions.

create_outline function generates an outline of objects in a frame.

create_skeleton function computes the skeleton of objects in a frame.

highlight_prominent_object function highlights the most prominent object in a frame.

detect_motion function detects motion between two consecutive frames based on their skeletons.

save_gif function saves a list of frames as a GIF file.

Figure 13 demonstrates python modules installed and imported based on the suggestion from ChatGPT.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 October 2023 doi:10.20944/preprints202310.1906.v1

Figure 13. Sentiment analysis of the libraries installed (left) vs libraries imported (right).

As can be seen from Figure 13 mainly commonly used libraries are mentioned in it. There are no rare or private modules and / or tools.

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* – [28] a technique renowned for its efficacy in image smoothing, *Canny Edge Detection* [29] – an advanced edge-detection method that outperforms basic thresholding, *Morphological Operations* [30] - 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, *Contour Approximation* [31] – a method that works by compressing horizontal, vertical, and diagonal segments, retaining only their endpoint.

Figure 13 illustrates the transformative impact of these methods. The left side of the figure presents the original image of a running cheetah, while the right side showcases the refined outline of the cheetah, achieved through the application of the methods mentioned above.

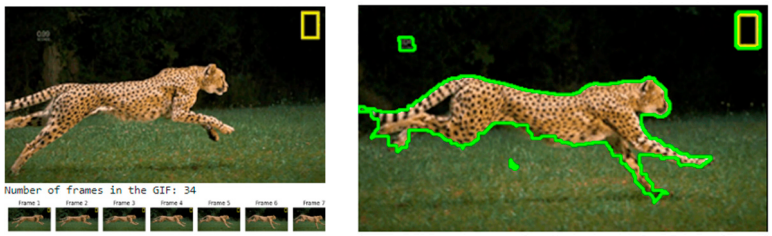

Figure 14. Splitting original image into frames (left) and creating its outline (right).

It appears that adding Contour Approximation significantly changes the outcome making the result less smooth what can be seen on Figure 15 (left). At the same time the amount of noise seems reduced compared to the result of three other methods combined.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 October 2023 doi:10.20944/preprints202310.1906.v1

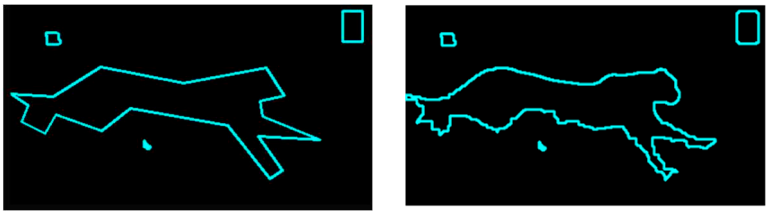

Figure 15. Outline with (left) vs without (right) Contour Approximation.

The pseudocode of Outline creation without Contour Approximation can be observed on 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) //
Extract contours
    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.
```

In the attempt to further reduce the noise, our own approach 'Cheetah in the circle' was applied. The most prominent object – the one with the biggest area was put in the circle and everything that appeared outside the circle was not considered to be a part of the motion. Figures 16.1 and Figure 16.2 highlighted the results.

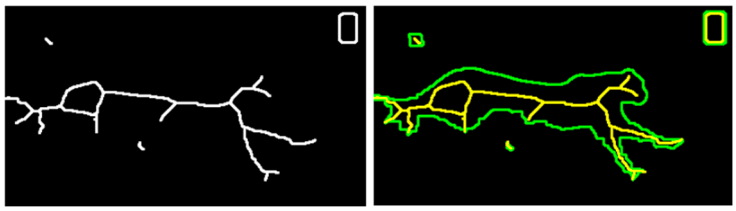

Figure 16.1. Skeleton without (left) vs with object outline (right).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 October 2023 doi:10.20944/preprints202310.1906.v1

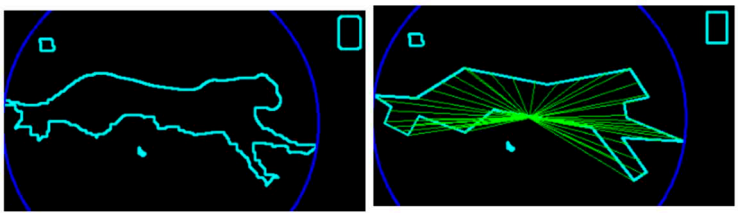

Figure 16.2. Identifying the prominent object surrounded by circle (left) and reduced noise revision (right).

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 plays role in the method as well and so the lines coming out fron the center of the circle to the outline of the object.

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.

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

As can be observed from Figure 17, the algorithm successfully works for many other images without any changes. At the same time, future work is required to make a motion detection algorithm of our own based on the custom Skeleton code and help from ChatGPT.

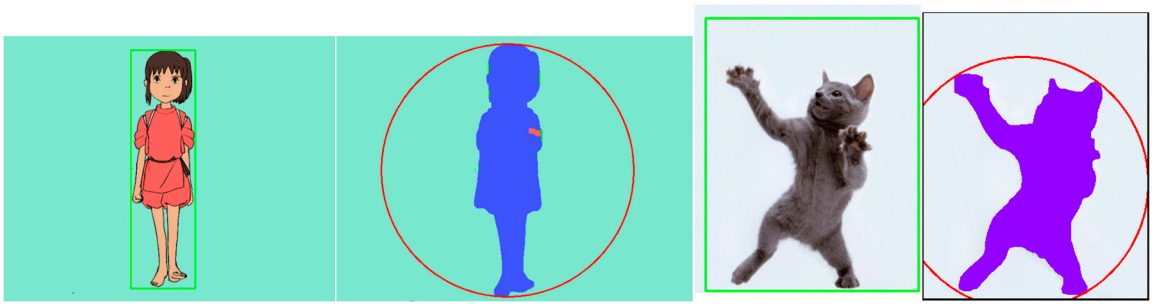

Figure 17. Algorithm validation using other images.

5. Discussion and Conclusion

During the study, we answered all the research questions we intended for this study:

The success of our code translation and app generation shows that LLMs, e.g., ChatGPT-4, Bard, and Bing, can indeed be used in this kind of application. In our study, we found that these models could understand the logic and structure of the MATLAB code and accurately translate it into Python code. They could also provide explanations and insights into the code's functionality. Furthermore,

they were able to identify and correct errors in both the MATLAB and Python code and improve the final code's quality based on their recommendations.

The results of the conversion performed by LLMs were superior to those of existing converter tools in terms of both accuracy and efficiency. The Python code produced by the LLMs was functional, accurate, and efficient. In contrast, existing converter tools often struggled with complex MATLAB code and were unable to translate it accurately. The LLMs were also more efficient, as they were able to translate the code in less time and with less human troubleshooting.

Furthermore, the process of MATLAB-to-Python conversion can be automated using the OpenAI API. However, there are potential challenges in this process. One challenge is that the API has limitations on the number of tokens that can be processed at once, which can make it difficult to translate large pieces of code. A potential solution to this challenge is to break the code into smaller chunks and translate them separately. Another challenge is that the API may not always produce accurate translations, especially for complex code. This can be mitigated by using the API in conjunction with manual review and testing to ensure the accuracy of the translated code. The last bottleneck is the commercial nature of the ChatGPT-4 and all such calls for conversion are not free. At this time the scalability of the project was not yet an issue.

It is highly feasible to make a Skeleton App with the help of LLM models once the M-to-PY translation is completed. In our study, we were able to successfully build a Skeleton App using the Python code produced by the LLM-based converter. The app was responsive, user-friendly, and able to accurately generate skeletons from images. This demonstrates that LLMs can not only assist in code translation but also the development of functional applications. The project of creating the app can be called Natural Language Coding (NLC), meaning that the model proposes what to do like which language, database, and architecture to pick for the app, but as well generated all required code and troubleshoots bugs and problems.

As the generation of functional programing code becomes more feasible and reliable, the coding efforts are shifted to testing efforts which can also be improved with the assistance of LLMs. We hope our research contributes to the validation of the capabilities of the latest LLMs and its usage in an app generation for image sketch abstractions towards 3D motion detections. We also hope that our generated and tested M-To-PY converter can help to bridge the gap between the scientific code in MATLAB and AI code in Python to facilitate scientific usage of AI technologies.

Supplementary Materials: The following supporting information can be downloaded at the website of this paper posted on Preprints.org.

Author Contributions: Conceptualization, Y.K. and Z.G.; methodology, Y.K. and Z.G; software, Z.G.; validation, Z.G., O.A. and J.J.L.; formal analysis, J.J.L. and K.L.; investigation, O.A.; resources, K.L.; data curation, X.X.; writing—original draft preparation, Y.K. and Z.G.; writing—review and editing, J.J.L., K.L, 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: Please add: This research was funded by the NSF, grants 1834620 and 2129795, and Kean University's Students Partnering with Faculty 2023 Summer Research Program (SPF).

Data Availability Statement: Data available on request due to privacy restrictions (personal nature of user-chatbot communication).

Conflicts of Interest: The authors declare no conflict 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. https://platform.openai.com/docs/quickstart. Accessed: 2023-08-14.
2. K. Leonard, G. Morin, S. Hahmann and A. Carlier, "A 2D shape structure for decomposition and part similarity," 2016 23rd International Conference on Pattern Recognition (ICPR), Cancun, Mexico, 2016, pp. 3216-3221, doi: 10.1109/ICPR.2016.7900130.

3. Javaid, Muhammad Adeel, Understanding Dijkstra's Algorithm (2013). http://dx.doi.org/10.2139/ssrn.2340905.
4. Dennis Layton. 2023. Open AI Code Interpreter. https://medium.com/@dlaytonj2/open-ai-code-interpreter-what-you-need-to-know-29c57085835e. Accessed: 2023-08-14.
5. OMPC. 2008. http://ompc.juricap.com/. Accessed: 2023-08-14.
6. Meta AI. 2020. Deep learning to translate between programming languages. 2023. https://ai.facebook.com/blog/deep-learning-to-translate-between-programming-languages/. Accessed: 2023-08-14.
7. Koziolek, H., Gruener, S., & Ashiwal, V. (2023). ChatGPT for PLC/DCS Control Logic Generation. arXiv:2305.15809.
8. Li, R., Pu, C., Fan, F., Tao, J., & Xiang, Y. (2023). Leveraging ChatGPT for Power System Programming Tasks. arXiv:2305.11202.
9. Tsai, M. L., Ong, C. W., & Chen, C. L. (2023). Exploring the use of large language models (LLMs) in chemical engineering education: Building core course problem models with Chat-GPT. Education for Chemical Engineers, Volume 44, July 2023, Pages 71-95.
10. Merow, C., Serra-Diaz, J. M., Enquist, B. J., & Wilson, A. M. (2023). AI chatbots can boost scientific coding. Nature Ecology & Evolution, 1-3. https://doi.org/10.1038/s41559-023-02063-3.
11. Kumar Y, Morreale P, Sorial P, Delgado J, Li JJ, Martins P. (2023). A Testing Framework for AI Linguistic Systems (testFAILS). Electronics. 2023, 12(14): 3095. https://doi.org/10.3390/electronics12143095.
12. Maurer, B. (2023) Geo P. Tech, AI Chatbot Geotechnical Engineer: How AI Language Models Like "ChatGPT" Could Change the Profession. https://doi.org/10.31224/2858.
13. Crokidakis, N., de Menezes, M. A., & Cajueiro, D. O. (2023). Questions of science: chatting with ChatGPT about complex systems. arXiv:2303.16870.
14. Nikolic, S. et. al. (2023). ChatGPT versus engineering education assessment: a multidisciplinary and multi-institutional benchmarking and analysis of this generative artificial intelligence tool to investigate assessment integrity. European Journal of Engineering Education, 1-56. https://doi.org/10.1080/03043797.2023.2213169.
15. Vercel. 2023. AI Code Translator. https://vercel.com/templates/next.js/ai-code-translator. Accessed: 2023-08-14.
16. CodeConvert. 2023. https://www.codeconvert.ai/. Accessed: 2023-08-14.
17. Ma, J., Ren, X., Li, H., Li, W., Tsviatkou, V. Y., & Boriskevich, A. A. (2023). Noise-against skeleton extraction framework and application on hand gesture recognition. IEEE Access, vol. 11, pp. 9547-9559, 2023, doi: 10.1109/ACCESS.2023.3240313.
18. Wang T, Yamakawa Y. Edge-Supervised Linear Object Skeletonization for High-Speed Camera. Sensors. 2023; 23(12):5721. https://doi.org/10.3390/s23125721.
19. Lovanshi, M., Tiwari, V. Human skeleton pose and spatio-temporal feature-based activity recognition using ST-GCN. Multimed Tools Appl (2023). https://doi.org/10.1007/s11042-023-16001-9.
20. Usmani, A., Siddiqui, N. & Islam, S. Skeleton joint trajectories based human activity recognition using deep RNN. Multimed Tools Appl (2023). https://doi.org/10.1007/s11042-023-15024-6.
21. Skeleton App. 2023. https://skeleton-app-65563db1db74.herokuapp.com/. Accessed: 2023-08-14.
22. Pankati Patel, Patricia Morreale, Yulia Kumar, Daehan Kwak, Jean Chu, Rose Garcia, Sabyatha Sathish and Margaret Burnett (2022) Embedding Equitable Design in the CS Computing Curricula. In proceedings of the SIGCSE 2023, https://doi.org/10.1145/3545947.3576278.
23. Pankati Patel, Patricia Morreale, Jean Chu, Yulia Kumar, Daehan Kwak, Rosalinda Garcia, Margaret Burnett (2022) Implementing Inclusive Software Design in the CS Curriculum. In Proceedings of the SIGCSE 2023, https://doi.org/10.1145/3545947.3573254.
24. Skeleton App Test cases. https://github.com/ykumar2020/SkeletonApp/blob/main/Test%20cases%20proposed%20by%20Bard%20and%20implemented%20by%20ChatGPT.pdf. Accessed: 2023-08-14.
25. Y. Kumar, P. Morreale, P. Sorial, J. Delgado, J. J. Li and P. Martins, "A Testing Framework for AI Linguistic Systems (testFAILS)," 2023 IEEE International Conference On Artificial Intelligence Testing (AITest), Athens, Greece, 2023, pp. 51-54, doi: 10.1109/AITest58265.2023.00017, https://ieeexplore.ieee.org/document/10229447.
26. A

27. B
28. Ritter, Frank (24 October 2013). "Smartphone-Kameras: Warum gute Fotos zu schießen nicht mehr ausreicht [Kommentar]". GIGA (in German). GIGA Television. Retrieved 20 September 2020. Bei Fotos, die in der Nacht entstanden sind, dominiert Pixelmatsch.
29. Canny, J., A Computational Approach To Edge Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986.
30. P Soille. "Morphological Image Analysis, Principles and Applications", 1999.
31. Chakraborty, D. "OpenCV Contour Approximation (cv2.approxPolyDP)," PyImageSearch, 2021, https://pyimagesearch.com/2021/10/06/opencv-contour-approximation/