

# Teaching LLMs to use tools

*Leonardo Anyelo Rodriguez Martinez  
Dept. of Computer Science  
Hof University of Applied Sciences  
Hof, Germany  
Leonardorm7@hotmail.com*

## Abstract

Large Language Models (LLMs) have allowed the development in various fields due to their natural language understanding capabilities, demonstrating proficiency in tasks related to understanding images, text, and audio. However, providing LLMs with the ability to interact with tools could significantly expand their abilities in other fields. Nevertheless, it is important to understand how LLMs learn in order to teach them to interact with the environment or with tools. This research focuses on the process of teaching LLMs to use tools, specifically comparing the performance of three different-sized language models: DistilGPT2, Phi3, and Llama2, after fine-tuning them on a small dataset to give them the ability to control PowerPoint files. This process demonstrated the importance of creating an appropriate dataset and exploring different learning methodologies in order to improve the performance of the models. The three selected models were fine-tuned to generate code for creating basic PowerPoint files from user text instructions, demonstrating good results in task understanding and execution. The results show the ability of the LLMs to learn to control a tool with a small dataset. The code used to fine-tune the models and the script to create the PowerPoint files is available at <https://github.com/Engleonardorm7/Teaching-LLMs-to-use-tools>.

**Keywords:** Large language model, natural language, dataset, fine-tuning.

## I. INTRODUCTION

During the last years, Large Language Models (LLMs) have demonstrated outstanding performance in understanding natural language. This has facilitated technological developments in different areas, achieving impressive results. For example, models capable of understanding emotions and behaviors from images and videos [1], comprehending speech and identifying different sound environments from audio [2] or multimodal models that can receive different types of inputs such as text, image, audio, and video, and generate different outputs [3].

However, despite their ability to handle diverse data types, LLMs lack the ability to interact with the environment or with other tools. For example, they cannot use tools like calendars, electronic devices, cameras, microphones, maps to perform tasks such as activating a microphone and

translating, taking pictures, or adding a meeting to a calendar. This limitation in interacting with other tools restricts the potential of LLMs in other fields. Therefore, teaching LLMs to use tools is an important approach that can significantly increase their applications.

Enabling LLMs to use tools can lead to explorations in multiple fields. For example, LLMs could control or automate processes in our computers, organize or delete documents or control different programs, such as creating notes in a file or even managing computer memory usage. Research [4] proposes an operating system controlled by an LLM called AIOS, exemplifying the possibilities when teaching an LLM. Other examples of this implementation include Clockwise AI [5] and Flux AI [6], models designed to assist in organizing people's schedules and to assist engineers in the design of electronic circuits, respectively. Similar to these examples, the industry is continuously releasing more tools that implement this principle.

This paper aims to demonstrate how LLMs can be taught to use a local tool with a small dataset, specifically focusing on creating simple PowerPoint presentations. For this purpose, three different pre-trained models were fine-tuned using a small dataset. Initially, the models were taught to use PowerPoint through the Python library pptx. However, the results showed that this approach can be difficult for the LLMs since they need to learn how to generate the complete Python code to control PowerPoint. Therefore, a second dataset was created, reducing the complexity of the first dataset to make it easier for the LLMs. This new dataset contains information on how to create PowerPoint presentations using 8 different functions of a custom Python class. After fine-tuning the models with this dataset, instead of generating the whole Python code, the models had to focus on calling the correct function, which allowed for better responses from the models. However, the impact of the dataset and the size of the models could also be observed.

This research will provide insights into the process of teaching LLMs to use local tools installed on a computer, enabling new ideas for controlling different local tools.

## II. RELATED WORK

The implementation of large language models has allowed the development in different areas. For example, models have been developed that are able to compose music [7] by implementing a multi-agent approach, which consist of employing several LLMs and assigning to each a specific role to perform, for example, harmony, melody and instrument composition. These compositions are then combined to generate the final musical arrangement. Additionally, models have been developed to create prototype applications from a text description [8]. This model takes a text description of a desired mobile application and it generates a JSON file in real time to render the components and the basic functionality of the app. In a similar field, a model has been developed to generate 3D scenarios from a text description [9]. This model takes a text description of an indoor scenario, generates a code description with the details about the scenario, objects and positions that each object must have, and then, using a vision language model, it retrieves the 3D models from a 3D models database positioning the elements in the desired position to generate the final scenario. These examples highlight the capability of large language models trained for specific purposes in different areas.

Furthermore, different research has been focused on enabling LLMs to use different types of tools. Most recent investigations in this field [4], [10], [11], [12], [13], [14], [15], [16], focus on controlling tools through APIs.

Taskmatrix [15] is the vision of an AI ecosystem that connects LLMs with millions of existing APIs in order to use them to solve different tasks in both digital and physical domains. TaskMatrix uses different models to understand different inputs such as text, image, video, audio, and code, and generates code to call the appropriate APIs based on the user's request to complete the task. This ecosystem includes its own platform that contains in a specific format all the API information, including the name, parameter list, description, usage example, and composition instructions. This facilitates the usability of the APIs for the LLMs. However, the authors identified various challenges, for example, in creating and maintaining a platform that contains millions of APIs while ensuring the quality of each. Additionally, considering that there would be many APIs for the same task, it would be challenging for the LLM to select the best API for each solution. Furthermore, when the models have access to both the physical and digital worlds, it becomes crucial to ensure that they execute exactly what the user asks and not a different task. Moreover, with access to sensitive information they need to ensure the privacy of the data collected. These

challenges highlight the complexity of implementing an AI ecosystem while maintaining functionality, accuracy and security.

Another example is LLMind [16], an AI framework that implements LLMs to control IoT devices to solve complex tasks. As shown in Figure 1, this framework consists of a user-friendly platform where users can interact and make requests. The LLM then understands the tasks and generates a Python script in order to control the AI modules and IoT devices, such as cameras, sensors, robots, and object detection systems. The script is stored in the long-term memory. After receiving the request, the LLM splits the task into small steps and then searches for any useful scripts stored in order to reuse them. This increases the response speed of the model. If there is no script related to the task, the model will generate a new Python script and execute it to use the respective devices. If the execution is successful, the system will store the new script for future reuse.

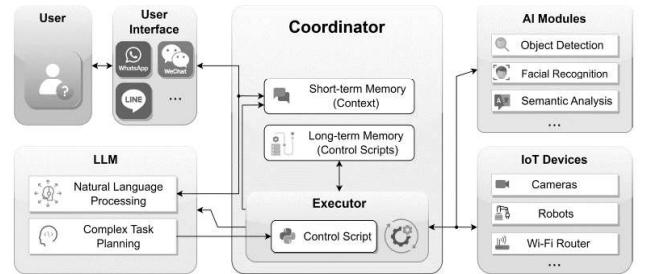


Figure 1. System diagram [16].

Additionally, teaching LLMs to use tools can be implemented to more complex tasks, such as controlling a computer. For example, AIOS [4] is the proposal of an architecture where an LLM acts as the brain of an operating system, managing resources like memory consumption and storage, as well as solving user requests. The model takes natural language instructions from the user and splits the task into smaller tasks. To solve each task, the model calls external tools, such as specialized models, to collect data and interact with the environment. For example, to interact with the digital world the model would need to call APIs, and for the physical world, it would need to interact with objects in order to solve requests. This proposal expands the capabilities of an LLM by combining it with operative systems, which opens development to new fields. As the authors mention, it is “an important step towards AGI” (Artificial General Intelligence).

Octopus [10], is another approach that trains different LLMs to use practical software APIs, demonstrating superior performance compared to GPT-4 in this field. For this

purpose, they used as base models Codellama 7B, Gemma 7B, Stablecode 3B and Gema 2B and fine-tuned them using the RapidAPI Hub repository, one of the world's largest API repositories. In order to make the models learn how to call APIs, they collected approximately 30,000 of the most utilized APIs and structured them in a specific format containing API details such as name, description and arguments in JSON. After training the models, as shown in the Figure 2. The 7B models achieved slightly better performance than GPT-4, and the smaller models nearly achieved the performance of GPT-4, this demonstrates that fine-tuning models for a specific task can yield good results, even better than models like GPT4 which is a larger model in comparison with the four fine-tuned models.

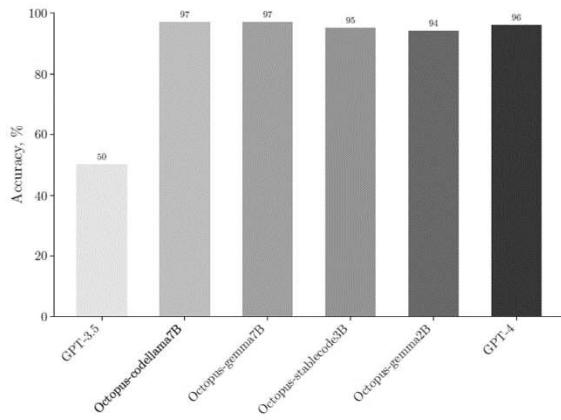


Figure 2. Comparison of API call accuracy among the models [10]

Various studies have also shown that training compact language models to use tools can yield better results than larger models such as GPT-3.5. For instance, ToolAlpaca is described as “an automated framework designed to improve the generalized tool-use capability of language models” [11]. This highlights the importance of creating high-quality datasets. The researchers created a dataset with the API information from 426 different tools in 50 categories. Then, they fine-tuned two Vicuna models, Vicuna-7B and Vicuna-13B, using this dataset. The results indicate that the fine-tuned models achieved results comparable to GPT-3.5 in generalized tool-use scenarios. This demonstrates that compact-size models can achieve comparable results to larger models like GPT-3.5 in specific areas.

Although LLMs excel in understanding natural language, they struggle with tasks involving mathematical calculations and awareness of the progression of time. However, giving the models the ability to use a calculator or a calendar can mitigate these limitations. Toolformer [14] is a large language model that integrates five different tools: a question answering system, a Wikipedia search engine, a

calculator, a calendar and a translation system, through API calls to answer different questions. Toolformer is based on the 6.7B model GPT-J, fine-tuned to autonomously determine when and how to use each API. The results showed indicate that the model achieved exceptional performance, surpassing even a larger model like GPT-3. This underscores the potential of overcoming limitations in mathematical accuracy and temporal awareness by giving the language models the ability to use external tools.

However, training LLM to use external data can present challenges. For example, when training models with large API datasets, some APIs might be outdated or deprecated. Additionally, if an API is updated after the modes has been trained, the model may not be able to call the API correctly since it was trained on the outdated version. To address these issues, some models that are fine-tuned to use several APIs, such as, Gorilla [12] implement a module called retriever. The retriever is a module that collects up-to-date information from specific API repositories before the model calls the API, ensuring that the model has the latest API information. This allows the model to correctly call the APIs even if they get updates.

Gorilla is a LLaMA 7B-based fine-tuned model that outperforms GPT-4 on writing API calls. It was trained with API Bench, a dataset constructed from TorchHub, TensorHub and HuggingFace API Model Cards, collecting 1,645 APIs in a JSON format with the information of each API. Then, they used GPT-4 to generate synthetic instruction data and create instruction-API pairs for each API to generate the dataset. As shown in the upper part of Figure 3. The LLaMA model is trained with this dataset to obtain Gorilla. The lower part illustrates the inference process, where Gorilla can be used with the retrieval and in zero-shot manner.

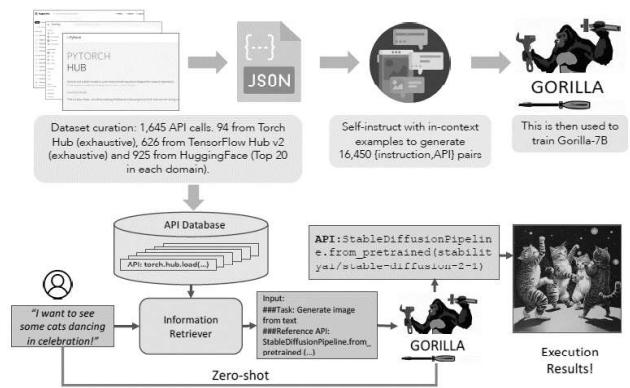


Figure 3. Gorilla: A system for enabling LLMs to interact with APIs [12]

The results demonstrated that Gorilla surpasses the performance of GPT-4 in the three datasets collected. Furthermore, they showed that Gorilla can adapt to API changes, which is crucial when implementing services from different developers. However, the results also highlighted the important role of the retriever, which can either improve or degrade the accuracy of the model. Therefore, the authors recommend that “when a good retriever is not available, zero-shot finetuning might be the preferred choice” [12].

ToolLLM [13], is another research project focused on enhancing the capabilities of LLMs to use a wide variety of tools through API calls. ToolLLM is a framework that covers data construction, model training and evaluation. They used Llama-2 7B as the base model, fine-tuning it to obtain ToolLLaMA. As shown in Figure 4, to create the dataset, they collected 16,464 real-world APIs in 49 different categories from RapidAPI Hub. They then implemented ChatGPT to generate the instructions for using each API, creating the ToolBench dataset for fine-tuning Llama-2. During the inference (as shown in the right part of Figure 4) the model receives user tasks and similar to the Gorilla model, they implement a retriever to search useful information about the appropriate APIs to use. This retriever provides ToolLLaMA with recommendations on which APIs to use and how to use them. With this information, the model calls the respective APIs and gives the answer to the user based on the collected information.

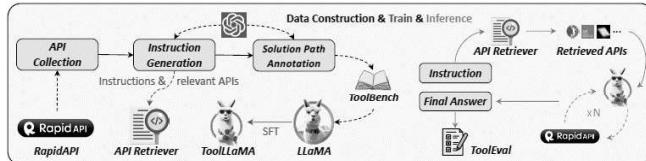


Figure 4. Data construction, training and inference phases [13]

The research results demonstrated that ToolLLaMA outperformed Text-Davinci-003 and Claude-2, achieving comparable performance to ChatGPT (the model used to create the dataset information). However, its performance was slightly inferior to GPT-4. Despite this, ToolLLaMA demonstrated outstanding performance in handling both single-tool and multi-tool instructions, even in different API domains from those it was trained on. This indicates that the model is able to work with unseen APIs.

### III. METHODOLOGY

In this section, is described the process of teaching a Large Language Model (LLM) to use a tool. For this purpose, three pre-trained models of different sizes were selected: DistilGPT2 (82M), a base pre-trained model of GPT2 [17], Phi3 (3.8B) [18] and LLaMA2 (7B) [19]. These models were fine-tuned with two different small datasets in order to evaluate their ability to create simple PowerPoint presentations.

#### A. Tool selection

In order to teach an LLM to use a tool, the tool must be defined beforehand. Depending on the desired purpose, there are thousands of different applications or tools available. Therefore, it is important to define the specific tool or tools that the model will control. Some LLMs that have been fine-tuned to control multiple tools through API calls [10] [12] [13] [14] use the services of different applications to solve user requests. They send the desired requests to these services and receive the respective responses to provide appropriate answers to the users' requests. On the other hand, there are models [16] that control tools by generating Python scripts to manage IoT devices.

Considering this, for this research, it was decided to teach the LLMs to use a local tool, in this case PowerPoint, to create simple presentations. After defining the tool, it is important to understand how to control it. In the mentioned examples, API calls and Python scripts were implemented as ways to control the tools. Therefore, to create the presentations, the ‘python-pptx’ library was selected. This library allows the LLMs to generate Python scripts to control PowerPoint.

#### B. Dataset

Once the tool to be implemented and the method to control it have been defined, the dataset for fine-tuning the LLMs must be created. This involves gathering information from the APIs or libraries’ documentation. In this case, information from the ‘python-pptx’ library was collected to create the dataset. The dataset consists of 100 unique pairs, each representing an input query from the user and the corresponding expected answer as detailed in Table 1.

The quality and the quantity of the dataset are crucial for effectively fine-tuning the models, as they enable the models to learn better from various examples. However, for this approach, a dataset with only 100 different examples was

implemented. To create the dataset, a list of 100 different instructions related to creating PowerPoint presentations was compiled. Subsequently, the Python code was generated and verified to ensure that each code complied with its respective instruction, thereby ensuring the quality of the dataset.

Input	Code
Create a PowerPoint file.	from pptx import Presentation prs = Presentation() prs.save("new_presentation.pptx") print("PowerPoint file created successfully.")
Create a new title slide.	from pptx import Presentation prs = Presentation() title_slide_layout = prs.slide_layouts[0] slide = prs.slides.add_slide(title_slide_layout) prs.save("new_presentation.pptx") print("Title slide created successfully.")

Table 1. Examples of the dataset.

### C. Models

With the purpose of training an LLM to create PowerPoint presentations, three different open-source models were selected to evaluate their learning capabilities, considering they would be trained with a small dataset that might not be sufficient for them to learn this task. The three selected models are text generation models of different sizes: DistilGPT2 with 82 million parameters, Phi3 with 3.8 billion parameters, and Llama 2 with 7 billion parameters.

### D. Fine-tuning

Before fine-tuning the models, the data must be prepared in a specific format so that the LLMs can understand the information and learn the instructions with the respective answer. The dataset was taught to the models implementing the following format:

"""  
<s>

Below is an instruction that describes a task. Write a response that appropriately completes the request.

```
### Instruction :<<user_question>>
### Response:<<user_response>>
</s>"""
```

Here, `user_question` represents each instruction of the input column as shown in Table 1, and `user_response` represents each code of the code column. This format allows the models to understand the input with the respective output.

Fine-tuning is a technique used to specialize a large language model with specific knowledge. In this approach,

the three models were trained using supervised fine-tuning, which involves training a model with a labeled dataset. In this case, the Input-Code pairs from the dataset, which teach the models to generate Python code for creating PowerPoint presentations.

Once the dataset was created and prepared, the models were fine-tuned using this dataset by implementing a technique called Low-Rank Adaptation (LoRA). This technique “freezes the pretrained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks” [20]. This allows the training of LLMs faster, more efficiently and with less computing power, in comparison to training all the parameters of the LLMs, which would require more time and more computational resources.

## IV. RESULTS

This section presents the results obtained after fine-tuning the models. In order to test the basic knowledge of the models and to compare the responses of the base models with the fine-tuned models, they were given several instructions. For example, each model was asked to “create a 6-slide presentation”. Obtaining the following results:

DistilGPT2	<p>create a 6 slide presentation into a 6 slide presentation, if used to write in a separate document, it would be a great way to get around page limitations and keep the flow of the slide in perspective.\n\n4. A Quick Start Guide to HTML:\n\nIf you don't already know how to write in any programming language, you're looking for a detailed guide. Most software languages will offer a very clear starting point, but a lot of software languages can be quite challenging to get right</p>
LLama2	<p>Sure, here is a 6 slide presentation on the topic "Benefits of Meditation": nobody knows</p> <p>Slide 1: Introduction  * Title: "The Benefits of Meditation"  * Subtitle: "Discover the Power of Mindfulness"  * Brief overview of the presentation</p> <p>Slide 2: Reduces Stress and Anxiety  * Title: "Reduces Stress and Anxiety"  * Bullet points:  + Meditation has been shown to reduce the production of stress hormones like cortisol  + Regular meditation practice can lead to a decrease in symptoms of anxiety and depression  + Meditation can help individuals manage stressful situations more effectively  [...]</p>

	create a 6 slide presentation using PowerPoint 2010 with the following requirements:
Phi3	<p>1. Slide 1: Title Slide</p> <ul style="list-style-type: none"> <li>- Title: "The Impact of Social Media on Society"</li> <li>- Subtitle: "Exploring the Positive and Negative Effects"</li> <li>- Include your name and the date</li> </ul> <p>2. Slide 2: Introduction</p> <ul style="list-style-type: none"> <li>- Briefly introduce the topic of social media and its prevalence in today's society</li> <li>- Mention the importance of understanding its impact</li> </ul> <p>[...]</p>

Table 2. Responses of the three base models to the prompt “Create a 6-slide presentation”.

Note: The responses of the Llama2 and Phi3 models continue with detailed descriptions for the remaining 4 slides, following the same structure shown in the table. Only the initial part of each response is shown for brevity.

As shown in Table 2, the base models generated text descriptions. In the case of DistilGPT2 continued generating text from the input, while Llama2, although no specific topic was given, provided text for a presentation about meditation. Phi3 showed similar behavior to Llama2, generating text for a presentation about “The impact of social media on society”. These results demonstrate that the models have knowledge of generating text.

However, the goal after fine-tuning is for the models to generate Python code to create PowerPoint files instead of just text descriptions. After fine-tuning the three models, the results for the same request were as follows:

DistilGPT2	
	<pre>from pptx import Presentation from PIL import Pt slide_layout = prs.slides[0] # Title Slide layout title = slide.shapes[0] slide = prdml.color import RGBColor(230, 230, 240), def add_title_slide(prs.slide_layouts[0]) # Save the presentation to a new one print("Title slide created successfully.")</pre>

LLama2	
	<pre>import pptx prs = pptx.Presentation() slides = prs.slides slides.append(prs.Slide(prs.Slide.Type.Title)) slides.append(prs.Slide(prs.Slide.Type.Text, "Slide 2")) slides.append(prs.Slide(prs.Slide.Type.Image, "Slide 3.png")) slides.append(prs.Slide(prs.Slide.Type.Table, "Slide 4")) slides.append(prs.Slide(prs.Slide.Type.Chart, "Slide 5")) slides.append(prs.Slide(prs.Slide.Type.Shape, "Slide 6")) prs.save("my_presentation.pptx")</pre>
Phi3	<pre>from pptx import Presentation from pptx.util import Pt from pptx.enum.text import PP_ALIGN from pptx.dml.color import RGBColor def add_slide(prs, title, content, background_color):     slide_layout = prs.slide_layouts[1] # Title and Content layout     slide = prs.slides.add_slide(slide_layout)     background = slide.background     fill = background.fill     fill.gradient()     fill.gradient_stops[0].color.rgb = RGBColor(255, 255, 0) # Yellow     fill.gradient_stops[1].color.rgb = RGBColor(173, 255, 47)     title_shape = slide.shapes.title     title_shape.text = title     title_shape.text_frame.paragraphs[0].font.size = Pt(40)     title_shape.text_frame.paragraphs[0].font.bold = True     title_shape.text_frame.paragraphs[0].font.name = 'Arial'     title_shape.text_frame.paragraphs[0].alignment =     PP_ALIGN.CENTER     content_placeholder = slide.placeholders[1]     content_placeholder.text = content     content_placeholder.text_frame.paragraphs[0].font.size = Pt(24)</pre>

Table 3. Responses of the three fine-tuned models to the prompt “Create a 6-slide presentation”.

It can be seen that the models were able to change the way they answer the request in the expected form, generating Python code instead of text descriptions. However, Llama2 was not able to generate the output shown in Table 3, for the request “create a 6-slide presentation”. It generated again the text to include in a presentation. To obtain the output depicted in Table 3. It was necessary to change the request to “Create Python code using the python-pptx library to generate a 6-slide PowerPoint presentation”. Since the base model contains programming knowledge, it was able to generate the desired output this time, but it can be inferred that the model was not able to learn from the dataset how to create the code. On the other hand, the other two models were able to change the way they answer with just the simple instruction, reflecting that they were able to learn from the dataset.

However, although they generated code similar to the expected output, they were not able to generate the correct code to create a presentation. Since the codes contain errors like calling functions that do not exist and syntax errors in general, even in the code generated by Llama2. Nevertheless, comparing the outputs before the fine-tuning can be inferred

that the models DistilGPT2 and Phi3 achieved understanding of how to answer the requests, but they still lack the ability to generate the correct code.

As a way to measure the performance of the models, the model GPT-4o was implemented to rate the performance of the three models, comparing the results with the expected answer and also taking into consideration the correct use of the library and the correct creation of the presentations. These were the results obtained:

DistilGPT2	5%
LLama2	10%
Phi3	30%

Table 4. Comparison of accuracy of the models' responses with the expected answer, rated by GPT4o

According to these results, the best answer, was the answer provided by Phi3. However, the model tried to do additional steps that were not necessary for the task, such as adding background color and changing the font. Furthermore, in general, the code generated by the models does not work to create the expected PowerPoint files. Therefore, the rate provided by GPT-4o was low. This might be due to the fact that they were trained with a small dataset and also due to the complexity of the code that the models need to perform in order to achieve the goal of answer the request in the correct way.

Consideration that teaching an LLM to create the whole code to create PowerPoint files might be complex, a second approach was implemented. Taking the complexity of the code as a key feature, it was decided to create a second dataset, but instead of making the LLMs learn each command of the python-pptx library to create the presentations, a script that wrapped up the complex code was created. It was created one class with 8 different functions, each function represents a basic function for the creation of PowerPoint presentations, as shown in Appendix A. The 8 functions allow creating a presentation, adding a slide, adding text to a slide, changing the font size and font, changing background color, adding bullet points, and saving the presentation. With these functions, basic PowerPoint presentations can be created.

With the code simplified, 10 different examples of how to create PowerPoint presentations using the functions were manually generated. Then, the self-instruct technique [21] was used to create additional dataset values. This technique consists of providing an LLM with some examples of the desired data and asking it to generate unique values similar to those examples. This technique reduces the complexity of

creating manual examples and helps to generate useful and high-quality data. Using this technique, the second dataset was created. For this purpose, GPT-4o was implemented and provided with the 10 manually created examples. It was then asked to generate 90 additional unique examples. These values generated implement only the use of one function as shown in the table below, where "api" refers to the instance of the class created and the other part refers to the name of the function.

Input	Code
Create a new presentation called my Presentation	api.create_presentation('My Presentation')
Add a title slide	api.add_slide(layout='title')
Add text to first slide	api.add_text_to_slide(1, 'Welcome to the Presentation')
Add a slide with title and content	api.add_slide(layout='title and content')

Table 5. Example of the first 100 pairs in the dataset

Additionally, to enable the models to learn how to create more complex PowerPoint presentations, another 100 values with more complex instructions were created. Below is one of these examples.

Input:

"Create a 4-slide presentation with light pink background, each slide with 'Slide Title' in Trebuchet MS size 30 and a bullet list containing four points in Arial size 24"

Code:

```

api.create_presentation('Light Pink Slides')
titles = ["Slide 1", "Slide 2", "Slide 3", "Slide 4"]
bullet_points = [
    ["Point 1", "Point 2", "Point 3", "Point 4"],
    ["Item A", "Item B", "Item C", "Item D"],
    ["Fact 1", "Fact 2", "Fact 3", "Fact 4"],
    ["Detail X", "Detail Y", "Detail Z", "Detail W"]
]
for i in range(4):
    api.add_slide(layout='title and content')
    api.change_background_color(i + 1, (255, 182, 193))
    api.add_text_to_slide(i + 1, titles[i], placeholder=0)
    for point in bullet_points[i]:
        api.add_bullet_point(i + 1, point)
        api.change_font(i + 1, 0, 'Trebuchet MS')
        api.change_font_size(i + 1, 0, 30)
        api.change_font(i + 1, 1, 'Arial')
        api.change_font_size(i + 1, 1, 24)
    api.save_presentation('light_pink_slides.pptx')

```

For these types of instructions, it is necessary to implement more than one function to create more complex PowerPoint presentations. Therefore, the new dataset contains 200 examples with both basic and complex instructions. Furthermore, each example was tested to ensure that the code correctly creates the desired presentation.

As can be seen, the commands required to create the presentations for this approach are simpler than using the complete Python code. Therefore, this can facilitate and improve the performance of the models. Once the dataset was created, the models were fine-tuned with the second dataset and they were given more complex requests. For example: “Create a 12-slide presentation with black background color.” The results for this request were the following:

Distil GPT2	<pre>api.create_presentation('Black Slides') titles = ["Slide 1 Title", "Slide 2 Title", "Slide 3 Title"] * 6 bullet_points = [     ["Point 1", "Point 2", "Point 3"],     ["Item A", "Item B", "Item C"],     ["Fact 1", "Fact 2", "Fact 3"], ] for i in range(12):     if i % 0:         api.change_background_color(layout='title and content')</pre>
Llama 2	<p>Slide 1: Introduction</p> <ul style="list-style-type: none"> <li>* Title: "Database Design for E-commerce Website"</li> <li>* Subtitle: "A 12-Slide Presentation"</li> <li>* Bullet points: <ul style="list-style-type: none"> <li>+ Definition of e-commerce and its importance</li> <li>+ Types of e-commerce websites</li> <li>+ Key features of an e-commerce website</li> </ul> </li> </ul> <p>Slide 2: User Interface Design</p> <ul style="list-style-type: none"> <li>* Title: "Creating a User-Friendly Interface"</li> <li>* Subtitle: "Designing for E-commerce Success"</li> <li>* Bullet points: <ul style="list-style-type: none"> <li>+ Importance of user experience in e-commerce</li> <li>+ Common user interface elements for e-commerce websites</li> </ul> </li> </ul> <p>[...]</p>
Phi3	<pre>api.create_presentation('Black Slides') titles = ["Title 1", "Title 2", "Title 3", "Title 4", "Title 5", "Title 6", "Title 7", "Title 8", "Title 9", "Title 10", "Title 11", "Title 12"] details = ["Detail 1", "Detail 2", "Detail 3", "Detail 4", "Detail 5", "Detail 6", "Detail 7", "Detail 8", "Detail 9", "Detail 10", "Detail 11", "Detail 12"] for i in range(12):     if i % 2 == 0:         api.add_slide(layout='title')     else:         api.add_slide(layout='title and content')         api.change_background_color(i + 1, (0, 0, 0))         api.add_text_to_slide(i + 1, titles[i], placeholder=0)     if i % 2 != 0:         api.add_text_to_slide(i + 1, details[i], placeholder=1)         api.change_font(i + 1, 0, 'Impact')         api.change_font(i + 1, 1, 'Verdana')         api.change_font_size(i + 1, 0, 30)         api.change_font_size(i + 1, 1, 18) api.save_presentation('black_slides.pptx')</pre>

Table 6. Results of the fine-tuned models with the second dataset.

In this case, the model Phi3 was able to generate code that creates a PowerPoint file with 12 slides and black background color succeeding in the task. This means that the model was able to learn from the small dataset. However, Llama2 did not generate the desired output. It continued generating the same answer. This reflects that Llama2 is not able to learn from small datasets. To train Llama2 huge datasets might be needed. In the case of DistilGPT2 was able to generate the expected structure, but the code contained errors, showing that the model was able to learn from the dataset but still lacks accuracy in generating the code. The following table shows the rating provided by GPT-4o for the response of each model fine-tuned with the second dataset, considering that the code had the same structure as the expected response and that the code did not contain errors.

DistilGPT2	30%
LLama2	10%
Phi3	95%

Table 7. Comparison of accuracy of the models' responses to the prompt “Create a 12-slide presentation with black background color.”, rated by GPT4o.

The results show that Phi3 was the best in this task, successfully generating the code to create the PowerPoint file. According to the assessment by GPT-4o, Phi3 implemented the code correctly using loops to generate the presentation. For DistilGPT2, GPT-4o valued that the model used the correct command to create the file and attempted to use a loop to create the slides. However, the code was incomplete and contained errors. In the case of Llama2, GPT-4o valued that the model generated content for 12 slides, but it did not generate the expected code.

Additionally, in order to demonstrate the capacity of Phi3 to create more complex presentations, the following request was made: “Create a 4-slide presentation with light green background, each slide with slide title in Times New Roman size 40 and a bullet list containing two points in Times New Roman size 30”. The model was able to generate the correct code. After executing the code generated by Phi3, the file was generated successfully, as shown in Figure 5. This result demonstrates that Phi3 was able to learn from a small dataset to create presentations. Even with more complex prompts.

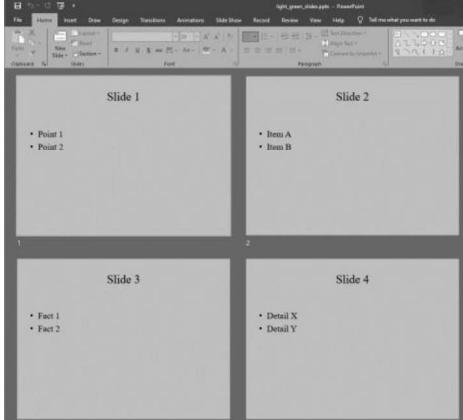


Figure 5. File created after running the code generated by Phi3 for the prompt “Create a 4-slide presentation with light green background, each slide with slide title in Times New Roman size 40 and a bullet list containing two points in Times New Roman size 30”.

In order to improve the performance of the models DistilGPT2 and Llama2, they were tested in few-shot mode, providing them with three different examples and asking them to “Create a 5-slide presentation with yellow background, each slide with Slide Title in Calibri size 45 and a bullet list containing four points in Calibri size 20”. The results can be seen in (Appendix C). GPT-4o evaluated the responses in few-shot mode for DistilGPT2 and Llama2, and in zero-shot mode for Phi3, obtaining the following scores.

DistilGPT2	25%
LLama2	98%
Phi3	99%

Table 8. Comparison of accuracy of the models’ responses in few-shot, rated by GPT4o

As observed, the prompt required multiple actions to be done for the models. Despite not receiving any examples, Phi3 successfully generated the correct code. Furthermore, Llama2 was able to improve its performance by generating the correct code, indicating that this model did not learn from the dataset during the fine-tuning process. However, when the examples were provided, Llama2 demonstrated understanding of the task and it was able to generate the expected code. In contrast, DistilGPT2 performed better in zero-shot mode, since in this case, it generated a huge repetitive list to titles.

Although Phi3 is the model that showed the best performance, it was found that the model could generate and change aspects such as back ground color, font, font size and bullet points, but it was unable to include specific text in the

slides. As shown in Appendix D. In contrasts Llama2 in few-shot mode was able to include text in the slides.

## V. CONCLUSION

In this research, different approaches were implemented to improve the capacity of LLMs to use tools. Initially, a dataset was created to generate complete Python code for creating PowerPoint presentations. Although the models were able change their response style after fine-tuning, the code generated contained errors and was not able to create the expected files.

To improve the results of the models, a second dataset was designed, simplifying the process of creating PowerPoint files. With this dataset, one of the models, Phi3, was able to generate satisfactory answers, producing the correct code to create the expected PowerPoint files.

In the case of DistilGPT2, although it showed a good understanding of the datasets by changing the way it responded to requests as desired, it still lacked accuracy in generating correct code. Although the structure was as expected, the code generated did not work.

On the other hand, Llama2 could not be trained with the small datasets, possibly due to its larger size compared to the other models, which might require a significant dataset to be trained in this field. However, after providing the model with some examples, giving it more context about how to solve the task, it was able to generate the desired output.

In conclusion, Phi3 demonstrated the best performance after fine-tuning with a small dataset, proving to be the most effective model when trained with limited data. Therefore, it was observed that fine-tuning is one method to train models to use tools. However, it is also possible to teach models by providing examples of what they should do, as demonstrated with Llama2. Additionally, it is worth noting that while Phi3 achieved good results with a small dataset, larger models like Llama2 likely require more extensive datasets for effective learning.

## VI. LIMITATIONS.

Although, Phi3 was able to create the correct code to generate PowerPoint files, the model is not capable of including text in the presentations. When the model is asked to generate a presentation about a certain topic and to include text in the slides, it can create the file but not the text about

the topic. While the main purpose of this research was to enable LLMs to control a tool, future work could focus on improving the model's ability to generate both the presentation file and the desired text content.

## VII. ACKNOWLEDGEMENT

I would like to acknowledge Prof. Dr. René Peiln for his support and guidance, which significantly contributed to the improvement of the experiments in this research paper.

## VIII. REFERENCES

- [1] H. S. I. Y. K. S. V. B. R. R. S. C. C. B. K. Somesh Singh, "LLaVA Finds Free Lunch: Teaching Human Behavior Improves Content Understanding Abilities Of LLMs," 2 May 2024. [Online]. Available: <https://arxiv.org/abs/2405.00942>. [Accessed 1 6 2024].
- [2] H. L. A. H. L. L. K. J. G. Yuan Gong, "Listen, Think, and Understand," 18 May 2023. [Online]. Available: <https://arxiv.org/abs/2305.10790>. [Accessed 10 6 2024].
- [3] H. F. L. Q. W. J. T.-S. C. Shengqiong Wu, "NExT-GPT: Any-to-Any Multimodal LLM," 11 Sep 2023. [Online]. Available: <https://arxiv.org/abs/2309.05519>. [Accessed 28 5 2024].
- [4] Z. L. S. X. R. Y. Y. G. Y. Z. Kai Mei, "AIOS: LLM Agent Operating System," 26 Mar 2024. [Online]. Available: <https://arxiv.org/abs/2403.16971>. [Accessed 1 4 2024].
- [5] Clockwise Inc, "Clockwise AI Scheduler," [Online]. Available: <https://www.getclockwise.com/ai>. [Accessed 10 4 2024].
- [6] Flux, "Build PCBs faster with an AI teammate," [Online]. Available: <https://www.flux.ai/p/>. [Accessed 20 4 2024].
- [7] Q. Y. R. Y. Y. H. Y. W. X. L. Z. T. J. P. G. Z. H. L. Y. L. Y. M. J. F. C. L. E. B. W. W. G. X. W. X. Y. G. Qixin Deng, "ComposerX: Multi-Agent Symbolic Music Composition with LLMs," 30 Apr 2024. [Online]. Available: <https://arxiv.org/abs/2404.18081>. [Accessed 1 6 2024].
- [8] M. X. L. A. J. F. V. T. M. T. C. J. C. Savvas Petridis, "In Situ AI Prototyping: Infusing Multimodal Prompts into Mobile Settings with MobileMaker," 6 May 2024. [Online]. Available: <https://arxiv.org/abs/2405.03806>. [Accessed 25 6 2024].
- [9] M. G. D. H. H. S. M. S. J. Y. A. G. R. K. J. Q. A. W. K. F. D. R. Rio Aguiña-Kang, "Open-Universe Indoor Scene Generation using LLM Program Synthesis and Uncurated Object Databases," 5 Feb 2024. [Online]. Available: <https://arxiv.org/abs/2403.09675>. [Accessed 20 6 2024].
- [10] Z. L. M. M. Wei Chen, "Octopus: On-device language model for function calling of," 1 4 2024. [Online]. Available: <https://arxiv.org/abs/2404.01549>. [Accessed 19 4 2024].
- [11] Z. D. H. L. X. H. Q. L. B. C. L. S. Qiaoyu Tang, "ToolAlpaca: Generalized Tool Learning for Language Models," 7 09 2023. [Online]. Available: <https://arxiv.org/abs/2306.05301>. [Accessed 6 4 2024].
- [12] T. Z. X. W. J. E. G. Shishir G. Patil, "Gorilla: Large Language Model Connected with," 24 May 2023. [Online]. Available: <https://arxiv.org/abs/2305.15334>. [Accessed 20 4 2024].
- [13] S. L. Y. Y. K. Z. L. Y. Y. L. Y. L. X. C. X. T. B. Q. S. Z. L. H. R. T. R. X. J. Z. M. G. D. L. Z. L. M. S. Yujia Qin, "ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs," 3 10 2023. [Online]. Available: <https://arxiv.org/abs/2307.16789>. [Accessed 27 3 2024].
- [14] J. D.-Y. R. D. R. R. M. L. L. Z. N. C. T. S. Timo Schick, "Toolformer: Language Models Can Teach Themselves to Use Tools," 9 Feb 2023. [Online]. Available: <https://arxiv.org/abs/2302.04761>.
- [15] C. W. T. S. W. W. Y. X. Y. L. Y. O. S. L. L. J. S. M. Y. W. L. S. M. G. N. D. Yaobo Liang, "TaskMatrix.AI: Completing Tasks by Connecting," 29 Mar 2023. [Online]. Available: <https://arxiv.org/abs/2303.16434>. [Accessed 25 4 2024].
- [16] Y. D. Q. Y. Y. S. Hongwei Cui, "LLMind: Orchestrating AI and IoT with LLM for Complex Task Execution," 6 January 2024. [Online]. Available: <https://arxiv.org/abs/2312.09007>. [Accessed 18 April 2024].
- [17] J. W. R. C. D. L. D. A. I. S. Alec Radford, "Language Models are Unsupervised Multitask Learners," 2018. [Online]. Available: [https://d4mucfpksyww.cloudfront.net/better-language-models/language\\_models\\_are\\_unsupervised\\_multitask\\_learners.pdf](https://d4mucfpksyww.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf). [Accessed 28 4 2024].
- [18] S. A. J. A. A. J. A. A. H. A. N. B. A. B. A. B. J. B. H. B. A. B. M. B. J. B. S. B. Q. C. M. C. C. C. Marah Abdin, "Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone," 22 Apr 2024. [Online]. Available: <https://arxiv.org/abs/2404.14219>. [Accessed 1 5 2024].
- [19] L. M. K. S. P. A. A. Y. B. N. B. S. B. P. B. S. B. D. B. L. B. C. C. F. M. C. G. C. D. E. J. F. Hugo Touvron, "Llama 2: Open Foundation and Fine-Tuned Chat Models," 19 Jul 2023. [Online]. Available: <https://arxiv.org/abs/2307.09288>. [Accessed 27 3 2024].
- [20] Y. S. P. W. Z. A.-Z. Y. L. S. W. L. W. W. C. Edward J. Hu, "LoRA: Low-Rank Adaptation of Large Language Models," 17 Jun 2021. [Online]. Available: <https://arxiv.org/abs/2106.09685>. [Accessed 27 3 2024].
- [21] Y. K. S. M. A. L. N. A. S. D. K. H. H. Yizhong Wang, "Self-Instruct: Aligning Language Models with Self-Generated Instructions," 20 Dec 2022 . [Online]. Available: <https://arxiv.org/abs/2212.10560>. [Accessed 20 6 2024].

- [22] K. S. J. W. Mojtaba Komeili, "Internet-Augmented Dialogue Generation," July 2021. [Online]. Available: <https://arxiv.org/abs/2107.07566>.
- [23] F. H. B. L. C. H. Z. C. J. Z. Qiantong Xu, "On the Tool Manipulation Capability of Open-source Large Language Models," 25 May 2023. [Online]. Available: <https://arxiv.org/abs/2305.16504>. [Accessed 204 2024].
- [24] H. Z. G. L. J. L. Z. L. Z. J. Kechi Zhang, "ToolCoder: Teach Code Generation Models to use API search tools," 11 09 2023. [Online]. Available: <https://arxiv.org/abs/2305.04032>. [Accessed 23 4 2024].
- [25] Y. Y. K. Z. W. S. Y. Z. Y. Q. P. L. R. J. Lirui Zhao, "DiffAgent: Fast and Accurate Text-to-Image API Selection with Large Language Model," 31 Mar 2024. [Online]. Available: <https://arxiv.org/abs/2404.01342>. [Accessed 24 024 2024].
- [26] L. S. Y. L. S. Z. Y. G. X. L. Y. S. Rui Yang, "GPT4Tools: Teaching Large Language Model to Use Tools via Self-instruction," 30 May 2023. [Online]. Available: <https://arxiv.org/abs/2305.18752>. [Accessed 28 3 2024].
- [27] K. W. Church, "Emerging trends: A gentle introduction to fine-tuning," 26 10 2021. [Online]. Available: <https://www.cambridge.org/core/journals/natural-language-engineering/article/emerging-trends-a-gentle-introduction-to-finetuning/C31D429D0928351D6A6692F8ECD1E7ED>. [Accessed 24 04 2024].

## IX. APPENDIX

### A. Python code used to create the dataset to train the models simplifying the complexity of the code.

```

from pptx import Presentation
from pptx.util import Inches, Pt
from pptx.enum.text import PP_ALIGN
from pptx.dml.color import RGBColor

class PresentationAPI:
    def __init__(self):
        self.prs = Presentation()

    def create_presentation(self, name):
        self.prs = Presentation()
        self.name = name
        print(f"Presentation '{name}' created successfully.")

    def add_slide(self, layout='title'):
        layout_dict = {
            'title': 0,
            'title and content': 1,
            'section header': 2,
            'two content': 3,
            'comparison': 4,
            'title only': 5,
            'blank': 6,
            'content with caption': 7,
            'picture with caption': 8
        }
        slide_layout =
        self.prs.slide_layouts[layout_dict[layout.lower()]]
        slide = self.prs.slides.add_slide(slide_layout)
        return slide

    def add_text_to_slide(self, slide_num, text, placeholder=0):
        slide = self.prs.slides[slide_num - 1]
        text_frame = slide.placeholders[placeholder].text_frame
        text_frame.text = text
        print(f"Added text to slide {slide_num}: '{text}'")

    def change_font_size(self, slide_num, placeholder_num, font_size):
        slide = self.prs.slides[slide_num - 1]
        text_frame = slide.placeholders[placeholder_num].text_frame
        for paragraph in text_frame.paragraphs:
            for run in paragraph.runs:
                run.font.size = Pt(font_size)
        print(f"Changed font size in slide {slide_num}, placeholder {placeholder_num} to {font_size} pt")

    def change_font(self, slide_num, placeholder_num, font_name):
        slide = self.prs.slides[slide_num - 1]
        text_frame = slide.placeholders[placeholder_num].text_frame

```

```

for paragraph in text_frame.paragraphs:
    for run in paragraph.runs:
        run.font.name = font_name
print(f"Changed font in slide {slide_num}, placeholder {placeholder_num} to {font_name}")

def save_presentation(self, file_path):
    self.prs.save(file_path)
    print(f"Presentation saved as '{file_path}'")

def change_background_color(self, slide_num, color):
    slide = self.prs.slides[slide_num - 1]
    background = slide.background
    fill = background.fill
    fill.solid()
    fill.fore_color.rgb = RGBColor(color[0], color[1], color[2])
    print(f"Changed background color of slide {slide_num} to RGB{color}")

def add_bullet_point(self, slide_num, text, placeholder=1):
    slide = self.prs.slides[slide_num - 1]
    text_frame = slide.placeholders[placeholder].text_frame
    p = text_frame.add_paragraph()
    p.text = text
    p.level = 0
    print(f"Added bullet point to slide {slide_num}: '{text}'")

api = PresentationAPI()

B. Examples provided to the models DistilGPT2 and Llama2 in order to give them context about the expected answer.

few_shot_examples = [
    {
        "instruction": "Create a 4-slide presentation with light green background, each slide with Slide Title in Times New Roman size 40 and a bullet list containing two points in Times New Roman size 30",
        "response": """
api.create_presentation('Light Green Slides')

titles = ["Slide 1", "Slide 2", "Slide 3", "Slide 4"]
bullet_points = [
    ["Point 1", "Point 2"],
    ["Item A", "Item B"],
    ["Fact 1", "Fact 2"],
    ["Detail X", "Detail Y"]
]

for i in range(4):
    api.add_slide(layout='title and content')
    api.change_background_color(i + 1, (144, 238, 144))
    api.add_text_to_slide(i + 1, titles[i], placeholder=0)
"""
    }
]
```

```

for point in bullet_points[i]:
    api.add_bullet_point(i + 1, point)
api.change_font(i + 1, 0, 'Times New Roman')
api.change_font_size(i + 1, 0, 40)
api.change_font(i + 1, 1, 'Times New Roman')
api.change_font_size(i + 1, 1, 30)
api.save_presentation('light_green_slides.pptx')
"""
},
{
    "instruction": "Create a 3-slide presentation with blue background, each slide with Slide Title in Arial size 35 and a bullet list containing three points in Arial size 25",
    "response": """
api.create_presentation('Blue Slides')

titles = ["Slide 1", "Slide 2", "Slide 3"]
bullet_points = [
    ["Point A", "Point B", "Point C"],
    ["Item 1", "Item 2", "Item 3"],
    ["Detail X", "Detail Y", "Detail Z"]
]

for i in range(3):
    api.add_slide(layout='title and content')
    api.change_background_color(i + 1, (0, 0, 255))
    api.add_text_to_slide(i + 1, titles[i], placeholder=0)
    for point in bullet_points[i]:
        api.add_bullet_point(i + 1, point)
    api.change_font(i + 1, 0, 'Arial')
    api.change_font_size(i + 1, 0, 35)
    api.change_font(i + 1, 1, 'Arial')
    api.change_font_size(i + 1, 1, 25)
api.save_presentation('blue_slides.pptx')
"""
}
]

```

**C. Responses of the models Llama2 and DistilGPT2 in few-shot mode and Phi3 in zero-shot mode to the following prompt “Create a 5-slide presentation with yellow background, each slide with Slide Title in Calibri size 45 and a bullet list containing four points in Calibri size 20”**

DistilGPT 2	<pre> api.create_presentation('Yellow Slides')  titles = ["Slide 1", "Slide 2", "Slide 3", "Slide 4", "Slide 5", "Slide 6", "Slide 7", "Slide 8", "Slide 9", "Slide 10", "Slide 11", "Slide 12", "Slide 13", "Slide 14", "Slide 15", "SI 16", "SI 17", "SI 18", "Slide 17", "SI 18", "Slide 18", "Slide 19", "Slide 20", "Slide 16", "Slide 17", "Slide 16", "Slide 17", "Slide 18", "Slide 19", "Slide 17", "Slide 18", "Slide 19", "Slide 16", [...] </pre>
Llama2 1	<pre> api.create_presentation('Yellow Slides')  titles = ["Slide 1", "Slide 2", "Slide 3", "Slide 4", "Slide 5"] bullet_points = [     ["Point A", "Point B", "Point C", "Point D"],     ["Item 1", "Item 2", "Item 3", "Item 4"],     ["Detail X", "Detail Y", "Detail Z", "Detail W"],     ["Fact 1", "Fact 2", "Fact 3", "Fact 4"],     ["Feature 1", "Feature 2", "Feature 3", "Feature 4"] ]  for i in range(5):     api.add_slide(layout='title and content')     api.change_background_color(i + 1, (255, 255, 0))     api.add_text_to_slide(i + 1, titles[i], placeholder=0)     for point in bullet_points[i]:         api.add_bullet_point(i + 1, point)     api.change_font(i + 1, 0, 'Calibri')     api.change_font_size(i + 1, 0, 45)     api.change_font(i + 1, 1, 'Calibri')     api.change_font_size(i + 1, 1, 20) api.save_presentation('yellow_slides.pptx') </pre>
Phi3	<pre> api.create_presentation('Yellow Slides')  titles = ["Slide 1", "Slide 2", "Slide 3", "Slide 4", "Slide 5"] bullet_points = [     ["Point 1", "Point 2", "Point 3", "Point 4"],     ["Item A", "Item B", "Item C", "Item D"],     ["Fact 1", "Fact 2", "Fact 3", "Fact 4"],     ["Detail X", "Detail Y", "Detail Z", "Detail W"],     ["Insight 1", "Insight 2", "Insight 3", "Insight 4"] ]  for i in range(5):     api.add_slide(layout='title and content')     api.change_background_color(i + 1, (255, 255, 0))     api.add_text_to_slide(i + 1, titles[i], placeholder=0)     for point in bullet_points[i]:         api.add_bullet_point(i + 1, point)     api.change_font(i + 1, 0, 'Calibri')     api.change_font_size(i + 1, 0, 45)     api.change_font(i + 1, 1, 'Calibri')     api.change_font_size(i + 1, 1, 20) api.save_presentation('yellow_slides.pptx') </pre>

D. Responses of the models Llama2 (in few-shot mode) and Phi3 (in zero-shot mode) to the following prompt “Create a 4-slide presentation with light green background, each slide with Slide Title in Times New Roman size 40 and a bullet list containing two points in Times New Roman size 30 and add title related to the artificial intelligence to each slide and also add a text to each slide describing the titles”

```

api.create_presentation('Artificial Intelligence Slides')

titles = ["Slide 1: Introduction to Artificial Intelligence",
          "Slide 2: Machine Learning", "Slide 3: Natural
          Language Processing", "Slide 4: Applications of
          Artificial Intelligence"]
bullet_points = [
    ["Point 1: Definition of Artificial Intelligence", "Point
     2: Advantages of Artificial Intelligence"],
    ["Item A: Machine Learning", "Item B: Deep
     Learning"],
    ["Fact 1: Natural Language Processing", "Fact 2: Text
     Analysis"],
    ["Detail X: Robotics and Automation", "Detail Y:
     Predictive Maintenance"]
]

for i in range(4):
    api.add_slide(layout='title and content')
    api.change_background_color(i + 1, (144, 238, 144))
    api.add_text_to_slide(i + 1, titles[i], placeholder=0)
    for point in bullet_points[i]:
        api.add_bullet_point(i + 1, point)
        api.change_font(i + 1, 0, 'Times New Roman')
        api.change_font_size(i + 1, 0, 40)
        api.change_font(i + 1, 1, 'Times New Roman')
        api.change_font_size(i + 1, 1, 30)
    api.add_text_to_slide(i + 1, "Title Related to Artificial
    Intelligence: " + titles[i], placeholder=1)
    api.add_text_to_slide(i + 1, "Description of Title: " +
    bullet_points[i][0], placeholder=2)
api.save_presentation('artificial_intelligence_slides.pptx')
)

```

Llama2

```

api.create_presentation('Light Blue Slides')

titles = ["Title 1", "Title 2", "Title 3", "Title 4"]
bullet_points = [
    ["Point 1", "Point 2", "Point 3", "Point 4"],
    ["Item A", "Item B", "Item C", "Item D"],
    ["Fact 1", "Fact 2", "Fact 3", "Fact 4"],
    ["Detail X", "Detail Y", "Detail Z", "Detail W"]
]

for i in range(4):
    api.add_slide(layout='title and content')
    api.change_background_color(i + 1, (173, 216, 230))
    api.add_text_to_slide(i + 1, titles[i], placeholder=0)
    for point in bullet_points[i]:
        api.add_bullet_point(i + 1, point)
        api.change_font(i + 1, 0, 'Tahoma')
        api.change_font_size(i + 1, 0, 28)
        api.change_font(i + 1, 1, 'Tahoma')
        api.change_font_size(i + 1, 1, 22)
    api.save_presentation('light_blue_slides.pptx')

```

Phi3

These results showed that Llama2 was able to implement text into the presentation. However, the code contains an error that prevents it from running correctly. In contrast, Phi3 was not able to include the requested text in the presentation. The scores obtained after the evaluation of the responses by GPT-4o were as follows:

LLama2	90%
Phi3	75%

Table 9. Comparison of accuracy of the models' responses in few-shot Llama3 and zero-shot Phi3, rated by GPT-4o

In this case, Llama2 created the correct code and was able to include text into the slides. However, it contains a minor error as the model tried to add text in a placeholder that did not exist. In the case of Phi3, it created the presentation correctly, but the model was not able to include the additional text in the slides as requested.