PPTC Benchmark: Evaluating Large Language Models for PowerPoint Task Completion

Yiduo Guo[1]*, **Zekai Zhang**[1]*, **Yaobo Liang**[2], **Dongyan Zhao**[1,3], **Nan Duan**[2]
[1]Wangxuan Institute of Computer Technology, Peking University
[2]Microsoft Research Asia
[3]National Key Laboratory of General Artificial Intelligence
yiduo@stu.pku.edu.cn, justinzzk@stu.pku.edu.cn, yaobo.liang@microsoft.com
zhaody@pku.edu.cn, nanduan@microsoft.com

Abstract

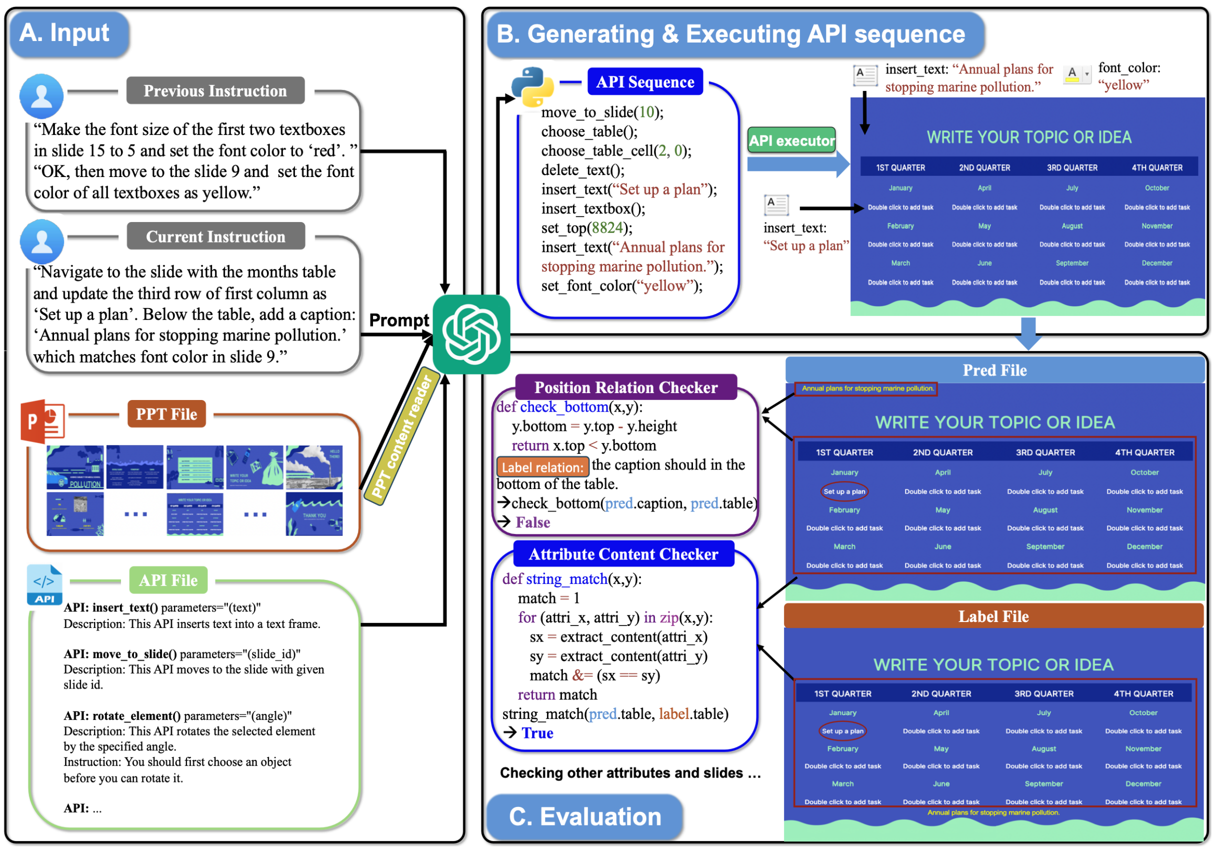
Recent evaluations of Large Language Models (LLMs) have centered around testing their zero-shot/few-shot capabilities for basic natural language tasks and their ability to translate instructions into tool APIs. However, the evaluation of LLMs utilizing complex tools to finish multi-turn, multi-modal instructions in a complex multi-modal environment has not been investigated. To address this gap, we introduce the PowerPoint Task Completion (PPTC) benchmark to assess LLMs' ability to create and edit PPT files based on user instructions. It contains 279 multi-turn sessions covering diverse topics and hundreds of instructions involving multi-modal operations. We also propose the PPTX-Match Evaluation System that evaluates if LLMs finish the instruction based on the prediction file rather than the label API sequence, thus it supports various LLM-generated API sequences. We measure 3 closed LLMs and 6 open-source LLMs. The results show that GPT-4 outperforms other LLMs with 75.1% accuracy in single-turn dialogue testing but faces challenges in completing entire sessions, achieving just 6% session accuracy. We find three main error causes in our benchmark: error accumulation in the multi-turn session, long PPT template processing, and multi-modality perception. These pose great challenges for future LLM and agent systems [1].

1 Introduction

Recent evaluation works for Large Language Models (e.g. ChatGPT and GPT-4 (OpenAI, 2023)) focus on their zero-shot/few-shot abilities on basic natural language tasks (Jiao et al., 2023; Zhong et al., 2023; Wang et al., 2023; Qin et al., 2023a) and their tool-use ability to generate APIs for solving user instructions, such as basic APIs like a calculator in tool transformer (Schick et al., 2023), RapidAPIs in ToolLLM (Qin et al., 2023c), and hugggingface APIs in Gorilla (Patil et al., 2023). However, these tool-use works emphasize the translation of natural language instructions into APIs and ignore the challenge of using APIs in the observation of complex multi-modal environments to finish user instructions. Also, their evaluation approach focuses on comparing the generated APIs with the label API sequence, assuming there's only one unique solution. This approach becomes impracticable in situations with multiple/unlimited correct solutions. To address these challenges, we introduce **P**ower-**P**oint **T**ask **C**ompletion (PPTC), a benchmark that measures LLMs' performance in creating and editing PPT file tasks based on user instructions. We choose PowerPoint as it includes various elements like textbox, table, and image and supports a wider range of APIs than Word and Excel.

Our benchmark has three distinctive features from other task completion benchmarks: (1) **Multi-turn dialogue with varying difficulty**. Our PPTC benchmark simulates the multi-turn dialogue session between the user and the LLM and contains 279 multi-turn sessions. Each multi-turn session in our benchmark includes 2 to 17 turns. Each turn consists of a user instruction that describes the user's needs, a feasible solution that provides the correct solution, and the resulting label output file. Some turns can be easily addressed using a single API, while over half of the instructions require multiple APIs for completion. We provide the LLM with a reference API file that contains all feasible APIs for selection. (2) **Multi-modality**. Finishing the instruction of our benchmark requires understanding the multi-modal PPT file content and using multi-modal API operations (e.g., PPTC has 268 image operation-related instructions). (3) **Evaluation based on the final status**: We propose the PPTX-Match Evaluation system to evaluate the LLM's outcome. To identify if the LLM completes the instruction, it checks the PPT file produced by

[1]Code: https://github.com/gydpku/PPTC



"Make the font size of the first two textboxes in slide 15 to 5 and set the font color to 'red'. "
"OK, then move to the slide 9 and set the font color of all textboxes as yellow."

"Navigate to the slide with the months table and update the third row of first column as 'Set up a plan'. Below the table, add a caption: 'Annual plans for stopping marine pollution.' which matches font color in slide 9."

API: insert_text() parameters="(text)"
Description: This API inserts text into a text frame.

API: move_to_slide() parameters="(slide_id)"
Description: This API moves to the slide with given slide id.

API: rotate_element() parameters="(angle)"
Description: This API rotates the selected element by the specified angle.
Instruction: You should first choose an object before you can rotate it.

API: ...

```
move_to_slide(10);
choose_table();
choose_table_cell(2, 0);
delete_text();
insert_text("Set up a plan");
insert_textbox();
set_top(8824);
insert_text("Annual plans for stopping marine pollution.");
set_font_color("yellow");
```

insert_text: "Set up a plan"
insert_text: "Annual plans for stopping marine pollution."
font_color: "yellow"

```
def check_bottom(x,y):
  y.bottom = y.top - y.height
  return x.top < y.bottom
```

Label relation: the caption should in the bottom of the table.

```
→check_bottom(pred.caption, pred.table)
→ False
```

```
def string_match(x,y):
  match = 1
  for (attri_x, attri_y) in zip(x,y):
    sx = extract_content(attri_x)
    sy = extract_content(attri_y)
    match &= (sx == sy)
  return match
string_match(pred.table, label.table)
→ True
```

Figure 1: We illustrate how LLMs complete one turn in a session. (A) To prompt the LLM, we provide it with the current instruction, previous instructions (dialogue history), PPT file content, and the API reference file. 'PPT reader' is a function that transforms the PPT file into the text-based format as the PPT file content. (B) The LLM then generates the API sequence and executes it to obtain the prediction PPT file. (C) We evaluate attributes and position relations in the prediction file.

executing the LLM-generated APIs rather than the LLM-generated APIs, thus all API sequences that lead to the correct final status are acceptable.

To finish the instruction, we use the current instruction, past turns' instructions (dialogue history), the PPT file content (specific environment information), and the reference API file as the input to prompt the LLM to generate an API sequence as the solution (See Figure 1 (A)). Then we use the API executor to execute the API sequence and return the user the resulting PPT file (See Figure 1 (B)). We name the resulting PPT file as the prediction file. In the evaluation step (See Figure 1 (C)), the PPTX-Match Evaluation system first uses the Python-PPTX library to extract all attributes from the prediction PPT file and the label output file. Then it uses the position relation checker to check if objects' positions conform to the label relation and the attribute content checker to check if the attribute's content is matched with the corresponding label attribute's content. The LLM correctly completes the current turn's instruction if all attributes of the file pass these tests.

We measure the performance of three closed-source LLMs (GPT-4, ChatGPT, and Davince-003) and six open-source LLMs (e.g., LLaMa-2) in our benchmark. We further test planning (e.g., CoT (Wei et al., 2022)) and content selection algorithms' performance based on GPT-4. Evaluation metrics include turn-based accuracy as the ratio of correctly completed turns to the total number of turns and session-based accuracy as the ratio of correctly completed sessions to the overall session count. Experiment results show that GPT-4 is the strongest LLM among all LLMs but still encounters challenges when completing entire multi-turn sessions. For example, although GPT-4 achieves 75.1% turn-based accuracy in the creating new PPT file task, it only achieves 22.7% session-based accuracy as errors made in previous turns. GPT-4 and other LLMs also struggle to process long PPT templates (complex file environment). For example, GPT-4 only achieves 38.1% turn-based accuracy in the editing task. We further find that GPT-4 struggles to finish instructions involving non-text modality operations, especially for position-related operations, such as 'Put object A on the top of the slide'. It only achieves 24% accuracy in these instructions.

In summary, this paper has the following contributions:

(1) We propose the PowerPoint Task Completion benchmark to measure LLM's task completion performance within the PowerPoint official software. This benchmark contains 279 multi-turn sessions with hundreds of multi-modal instructions in the complex multi-modal environment.

(2) We propose the PPTX-evaluation system to automatically measure LLMs' performance in our benchmark. We test 3 closed-source LLMs and 7 open-source LLMs and find that GPT-4 is the strongest LLM among all LLMs.

(3) We further analyze LLMs in our benchmarks and find three key error factors: error accumulation in the session, long PPT template processing, and multi-modality perception. These findings pose significant challenges for future LLMs and LLM-based systems.

2 PPTC Benchmark

In this section, we introduce our Power-Point Task Completion (PPTC) benchmark, including the overview of our benchmark, its collection and validation process, and the PPTX-Match Evaluation System for evaluation. We further analyze the statistics information of our benchmark.

2.1 Benchmark Overview

Benchmark components Our benchmark focuses on two basic tasks within PowerPoint: creating the new PPT file and editing the existing long PPT template for measuring long PPT Content understanding. We have gathered 229 multi-turn dialogue sessions for creating the new PPT file and 50 sessions for editing existing templates. Each multi-turn session includes 2 to 17 turns. Each turn comprises three parts: (1) the user instruction (2) the label output file as the ground truth (3) one feasible API sequence for finishing the instruction. Our benchmark also contains an API reference file that includes 49 feasible APIs for various operations and can complete all instructions in our benchmark. For each API, we describe its functionality and arguments and provide usage guidelines. For complex APIs, we also offer example cases. We also provide a PPT reader function and an API executor for LLMs to process the multi-modal PPT file and execute the API sequence, respectively. The details of all APIs, PPT reader, and API executor are in Appendix A.

Task description To complete the instruction in one turn, in general, the AI assistant must comprehend the user's current and prior instructions for context. It should also analyze the content of the PPT file to identify relevant objects mentioned in the instruction. Additionally, it needs to select appropriate APIs from a reference API file to achieve the user's goals. So we use these as the input of the AI assistant and it should output an API sequence as the solution. Then, it executes this API sequence and provides the user with the resulting PPT file as its response (See the whole process in Figure 1).

2.2 Benchmark Collection

Design Principles We follow these principles to design our benchmark: (1) Multi-turn instructions: One session in our benchmark should contain multi-turn instructions to finish the user's complex need. (2) Instructions of varying difficulty: Some instructions can be achieved with a single API, while others necessitate a sequence of APIs for successful completion. (3) Diverse multimodal operations: User instructions should cover a wide range of operations on PPT, such as text-related, image-related, and position-related APIs. (4) Topic Consistency: The dialogue in a session should center around the session topic. Each user instruction in a session aligns closely with the previous instructions (the context), ensuring a coherent and contextually relevant dialogue flow. (5) Practicability First: The session topic and specific instructions should simulate the user's need in real world

Benchmark Collection and Validation To collect user instructions, we engage 6 skilled crowd workers who craft instructions in accordance with the principles we've outlined. To achieve practicability first, we request crowd workers to write instructions based on their actual PowerPoint experience. On each session, the workers are asked to first find and list a practicable session topic. For the editing PPT template task, the topic must based on the template file background and is practicable to the template[2]. To achieve multi-instructions and topic consistency, the workers write instructions

[2]We collect 50 PPT templates from the SlidesCarnival website (`https://www.slidescarnival.com/`).

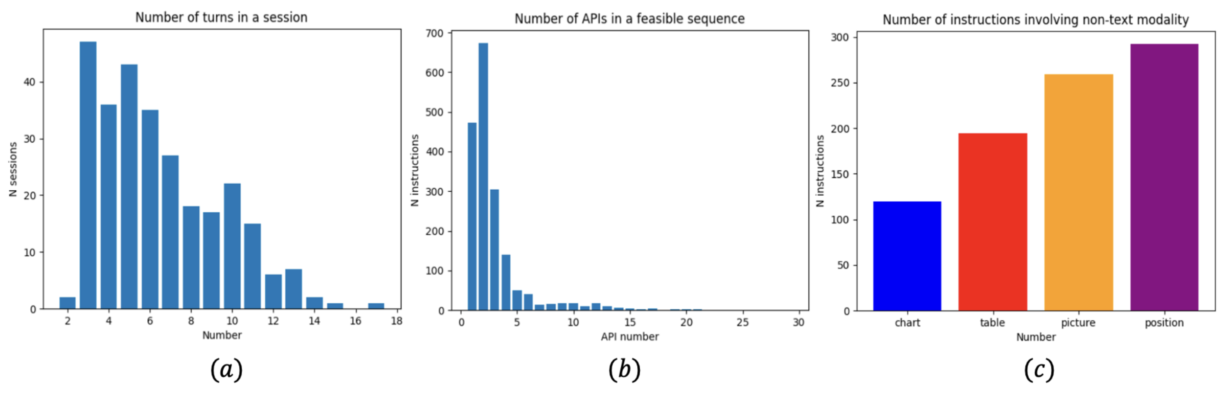


Figure 2: Statistics for PPTC. a) Session turn number distribution. b) Instruction API number distribution (tokens). c) Distribution of instructions involving Chart, Table, Picture, and Position. Instructions involving ’Position’ need the system to conduct position-related operations based on the understanding of spatial information. Note that one instruction may involve multiple different modalities.

step by step and make them consistent with the topic. To achieve diverse multi-operations, we ask them not to write session that only involves a single modality operation.

Then we ask the seventh worker to write the feasible API sequence with minimal API usage for each instruction. Next, the workers create the PPT label file by using the provided API sequence. More details and steps for the collection and validation process are in Appendix B.

2.3 PPTX-Match Evaluation System

We design the PPTX-Match Evaluation System to evaluate LLMs’ performance on the PPTC benchmark. It’s based on the PPT file (final status). To judge if the LLM has successfully completed the user instruction in one turn, it compares all non-position attributes in the LLM’s prediction file with those in the label file and verifies if objects follow the correct position relation. The detailed steps are in Appendix C.

2.4 Benchmark Statistics Analysis

To understand the properties of PPTC, we analyze the instructions and APIs and report statistics about the PPTC benchmark in Figure 2.

The number of turns in a session The session turn number distribution (Figure 2 (a)), measured as the number of turns in a session, shows that almost all sessions have at least 3 turns (between 3 and 13 turns for the 5th to 95th percentile, respectively). The longest session has 17 turns, which is very challenging as the errors made in previous turns can influence the completion of the current instruction.

Diffculty varies in APIs number The number of APIs in a sequence falls between 1 and 5 for the 5th to 95th percentile (Figure 2 (b)), respectively, shows that our instructions vary from a simple one that can be finished by one API to a complex instruction that requires multiple APIs. The longest API sequence consists of 29 APIs. Generating long API sequences is very challenging as the LLM needs to understand sub-goals in the complex instruction, select appropriate APIs, and generate APIs in a reliable order.

Rich multi-modal instructions Our benchmark has hundreds of instructions that involve multi-modalities content (Figure 2 (c)). The "chart" modality has the fewest instructions, with 120, while the "position" modality has the most, with 292 instructions. To finish these instructions, LLMs need to employ related-modal APIs based on the understanding of multi-modal file content.

3 Algorithms

In this section, we introduce the algorithms we considered to enhance the LLM’s performance in our benchmark. These algorithms can be categorized into two approaches: planning algorithms that help the LLM in decomposing the user instruction and solving it step by step and selection algorithms that assist the LLM in choosing important environmental information or APIs.

3.1 Planning Algorithms

Complex user instructions often require multiple intermediate steps to complete. We mainly consider two planning algorithms:

Zero-shot-CoT (Kojima et al., 2022) enables LLMs to autonomously generate intermediate

Inference prompt in PPTC

(**Task instruction**) You are an AI assistant to help the user to operate PowerPoint and edit the contents.
Give you the user instruction:<Current user instruction>, you can complete it based on the following APIs and PPT file content. Current you are at page <Page id>. Please finish the user instruction with the functions you have. Don't generate instructions beyond what the user has instructed. Don't guess what the user may instruct in the next step and generete API for them. Don't use python loop to call API. You can only call API once in one line. If the user does not specify the page to be modified, you can directly start using the APIs without having to navigate to other pages.
You need to generate code which can finish the user instruction. The multiple lines of code should be surrounded by <code> and </code> such as: <code> API(); API(); </code>
For example, if the user instruction is "create a slide", then the answer should be:
<code> create_slide(); </code>

(**API file**) Now, you have access to a list of PowerPoint APIs with the following functions: <APIs and their descriptions>
(e.g.,API(name="set_width", parameters="(width)",
description="This API sets the width of the selected object.",
parameter_description="It takes one parameter 'width', the width of an object in centimeters as float.",
composition_instruction="You should first choose an object before you can change the width of it.",
api_desc="width of picture and shapes"))

(**PPT file content**) All the PPT contents are:
<Begin of PPT>
Turn-based: <Parsed PPT file content of the label PPT file of the previous turns>
Session-based: <Parsed PPT file content of the LLM prediction file of the previous turns>
<End of PPT>

(**Dialogue history**)
¬User¬: Hello!
¬AI¬: Hi there! How can I help you?
¬User¬: <the first instruction>
¬AI¬:
Turn-based: <the correct feasible API sequence>,
Session-based: <the LLM-generated API sequence>
...
¬User¬: <Current user instruction>. Surrounding your answer with <code> and </code>.
¬AI¬:

Figure 3: The inference prompt we used in both turn-based and session-based evaluation settings. In the turn-based evaluation, we assess the LLM's performance for the current turn and assume the LLM has correctly finished previous turns. We then use feasible API sequences of previous turns as the AI response in the dialogue history and parse the label file of previous turns as the PPT file content. In the session-based evaluation, we evaluate the completion of the entire session and do not assume the LLM has correctly finished previous turns. We use the LLM's generated API sequences as the response and parsed the LLM prediction file as the PPT file content.

reasoning processes for complex instruction by prompting LLMs to "Let's think step by step".

Tree of Thoughts (ToT) (Yao et al., 2023) enables LLMs to follow tree-like reasoning paths, where each tree node represents a thinking state. It leverages LLMs to generate evaluations or votes on different thoughts.

3.2 Selection Algorithms

Combining the whole PPT file and the whole API file into the LLM's input can result in redundant information and filtering out them would improve the efficiency of the LLM. In this context, we primarily focus on two algorithms for selecting the relevant PPT file content and helpful APIs, respectively:

Content Selection algorithm Firstly, we extract all shapes of the PPT file by Python-PPTX. Next, we prompt the LLM to select the shapes for completing the user's instruction. We show the prompt in Figure 9 of Appendix D, in which we add three demonstration examples to guide the LLM to do selection. In this algorithm, we replace the whole PPT file with the selected shapes when prompting the LLM to generate the API sequence.

API Selection algorithm The API selection algorithm is based on the embedding similarity to select the most relevant APIs for user instructions. Specifically, we use the text embedding API to get the embeddings of all API descriptions and the current user instruction. Next, we compute the cosine similarity between the instruction embedding and each API description's embedding and rank them based on the similarity score. In this algorithm, we replace the whole reference API file with the top k APIs when prompting the LLM to generate the API sequence.

4 Experiments

4.1 Large Language Models Selected for Evaluation

We assess different cutting-edge large language models using our benchmark. These chosen models showcase a wide array of capabilities and are highly regarded in the field. The evaluated large language models include 3 closed-source LLMs:*GPT-4* (OpenAI, 2023), *ChatGPT*, and *Text-Davinci-003* (Brown et al., 2020). We also consider 7 open-source LLMs: *LLaMa-2-Chat* (Touvron et al., 2023), *Baichuan-Chat*, *Baichuan-2-Chat* (Yang et al., 2023), *WizardLM v1.2* (Xu et al., 2023a), *Vicuna v1.5 (16k)* (Chiang et al., 2023), and *Code-LLaMa-instruct* (Chiang et al., 2023), and *Mistral-instruct* (Jiang et al., 2023). The detailed introduction of these LLMs is in Appendix E.

4.2 Experimental Setup

In this section, we provide an overview of the experimental setup utilized to assess the performance of LLMs on our PPTC benchmark.

4.2.1 Turn-Based and Session-Based Evaluations

We consider two performance evaluation approaches in our benchmark: turn-based and session-based evaluations. For the turn-based evaluation, we measure the LLM's ability to finish a single turn. For the session-based evaluation, we measure the LLM's ability to finish a session containing multiple turns. One core difference is that the turn-based evaluation assumes that the previous turns have been correctly finished but the session-based evaluation does not. We put more details and illustrate the prompts for the two evaluations in Appendix F.

Metrics For turn-based evaluation, we report the turn-based accuracy as the ratio of the number of successfully finished turns to the total number of turns. We also report the average token number of the input of one turn and the average API number for finishing one turn as the cost measurement. For session-based evaluation, we report the session-based accuracy as the ratio of the number of successfully finished sessions to the total number of sessions. We also report the average value of the token number of all inputs in one session and the average API number required to complete one session as the cost measurement.

4.3 Implementation Details

All closed-LLMs experiments were conducted using the respective language models' API provided by Azure OpenAI Service. For open-source LLMs, we choose their chat/instructed version with 13 billion parameters model. For the zero-shot CoT method, we add the sentence 'Let's think step by step' after the dialogue history of the prompt. Our inference prompts are in Figure 3. We run the four algorithm methods based on the GPT-4 model. More details are in Appendix G

4.4 Main results

We report the results of LLMs in both turn-based and session-based evaluations in Table 1 and 2. From the results, we highlight the following key findings.

(1) Superior Performance of GPT-4: GPT-4 consistently outperforms other closed-source and open-source LLMs in both two tasks. For example, GPT-4 achieves 75.1% turn-based accuracy in the creating new PPT file task, demonstrating its strong capability to finish one turn of the user instruction. GPT-4 also has a lower API cost compared to other closed-source LLMs since its precise API usage. GPT-4 incurs the highest token expense when editing PPT templates. That is because its higher token limit than other LLMs allows us to input more PPT template content.

(2) Code continual pre-training and further instruction finetuning can boost open-source LLMs' performance.: Current open-source LLMs struggle to match the performance of closed-source LLMs in Table 1. For example, LLaMa-2-chat only achieves 16.2% turn-based accuracy in the creating PPT slide task. Code continual pretraining (Code-LLaMa) and instruction fine-tuning based on LLaMa-2 (WizardLM and Vicuna) can further improve LLaMa-2 potential PPT task completion performance on our benchmark. For example, Code-LLaMa improves LLaMa-2's turn-based accuracy in the creating new PPT slide task by 20.4 %.

(3) Planning and selection algorithms can improve LLMs' turn-based performance From Table 2, we observe that the planning algorithms (CoT

Models and Methods	Creating new PPT						Editing PPT template					
	Turn-based			Session-based			Turn-based			Session-based		
	Accuracy	Avg token	Avg API	Accuracy	Avg token	Avg API	Accuracy	Avg token	Avg API	Accuracy	Avg token	Avg API
TD-003	72.6	2.8k	3.0	12.7	20.8k	23.9	24.4	2.9k	8.1	4.0	13.2k	26.6
ChatGPT	70.6	2.9k	3.2	12.7	20.0k	23.4	26.3	4.1k	7.9	2.0	9.2k	22.9
GPT-4	75.1	2.9k	2.9	22.7	20.8k	22.4	38.1	7.5k	7.8	6.0	24.1k	24.7
LLaMa-2	16.4	2.8k	3.9	3.4	21.6k	24.7	8.7	2.2k	7.2	0.0	9.5k	15.6
Code-LLaMa	36.8	2.8k	3.4	0.0	20.7k	32.1	18.7	3k	7.3	2.0	9.6k	22.6
WizardLM	23.9	1.3k	3.3	4.3	12.5k	22.4	10.0	1.3k	5.7	0.0	4.3k	16.5
Vicuna-v1.5	24.3	1.3k	3.9	2.2	11.0k	33.7	6.8	1.3k	6.7	0.0	4.3k	22.7
Baichuan	15.5	1.3k	9.8	0.0	10.9k	44.7	4.4	1.3k	9.6	0.0	4.3k	24.3
Baichuan-2	16.3	1.3k	9.1	3.6	11.6k	48.9	8.7	1.3k	9.2	0.0	4.2k	22.3
Mistral-Instruct	14.1	1.3k	12.1	0.0	11.6k	73.5	12.5	1.3k	17.9	2.0	5.1k	41.6

Table 1: We report the results of LLMs and methods based on GPT-4 in this table.' TD-003' is the Text-Davinci-003 model. We directly use the prompts in Figure **??** to prompt LLMs to generate the API sequence.

Models and Methods	Creating new PPT file						Editing PPT template					
	Turn-based			Session-based			Turn-based			Session-based		
	Accuracy	Avg token	Avg API	Accuracy	Avg token	Avg API	Accuracy	Avg token	Avg API	Accuracy	Avg token	Avg API
GPT-4	75.1	2.9k	2.9	22.7	20.8k	22.4	38.1	7.5k	7.8	6.0	24.1k	24.7
GPT-4+CoT	77.0	2.9k	3.1	23.1	20.8k	22.7	40.6	7.5k	8.0	6.0	24.1k	25.2
GPT-4+ToT	76.5	20.8k	3.0	21.8	146.4k	22.6	40.6	81k	7.6	4.0	256.8k	24.0
GPT-4+Content selection	77.5	3.4k	3.0	21.8	24.5k	22.0	43.1	5.8k	8.0	4.0	18.7k	25.2
GPT-4+API selection	76.4	1.5k	2.9	18.8	10.6k	21.3	38.1	7k	8.0	10.0	22.4k	25.8

Table 2: We report the results of GPT-4 and algorithms based on the GPT-4 model. 'CoT' and 'ToT' are the chain of thought and tree of thought algorithms.

and ToT) can further improve the turn-based performance of GPT-4 by 1∼2 percent. However, we find that the more complex ToT algorithm does not outperform the zero-shot CoT algorithm with a 5∼10 times token cost. Content and API selection algorithms can further improve the turn-based performance of GPT-4 by 1∼ 5 percent. That is because they reduce the task difficulty by filtering irrelevant PPT content/APIs in the input prompt. The API selection algorithm also reduces the average token cost by reducing the number of APIs listed in the prompt.

4.5 Three challenges in our PPTC benchmark

From the result Table 1 and Figure 4. we highlight the following three key challenges.

(1) Error accumulation makes LLMs performance poor in finishing the entire multi-turn session.: The performance of all LLMs in handling sessions consisting of multiple turns is notably poor. GPT-4, achieves only a 22.7% session-based accuracy for the "creating new PPT file" task and a mere 6.0% session-based accuracy for the "editing PPT template" task. The session-based evaluation is challenging since errors made in previous turns make the LLM fail to finish the session and also influence the completion of the current turn. Current planning and selection algorithms usually fail to improve session-based accuracy markedly. In some cases, they can even make the performance worse.

(2) LLMs perform badly in processing long PPT template: Current LLMs' performance in the editing PPT temples task is pretty poor. For example, the strongest GPT-4 only achieves 38.1% turn-based accuracy and 6.0% session-based accuracy in this task. The content selection algorithm can partially solve this challenge by filtering out irrelevant file content, but GPT-4 with it still only achieves 43.1% turn-based accuracy. For open-source LLMs, there's a risk of information loss due to token limitations (typically 2∼4K tokens limit), which often require truncating lengthy PPT content. When it comes to session-based performance, the accuracy remains nearly zero. That means current LLMs (e.g., GPT-4 and LLaMa-2) still struggle to handle complex and lengthy PPT templates.

(3) Multi-modal instructions increase the LLM's failure rate significantly. To assess LLMs' task completion performance for instructions involving multi-modal operations (Table, Chart, Picture, Position, and text), we calculate the average accuracy of GPT-4 for each modality by dividing the number of correctly completed instructions within each modality by the total number of instructions involving that modality's operation. The results are presented in Figure 4 (a). From the figure, we observe that GPT-4's performance becomes poorer when processing structured data (Chart and Table) rather than text, with 12.4% and 16.2% lower accuracy. Instructions involving picture-related operations pose a greater challenge

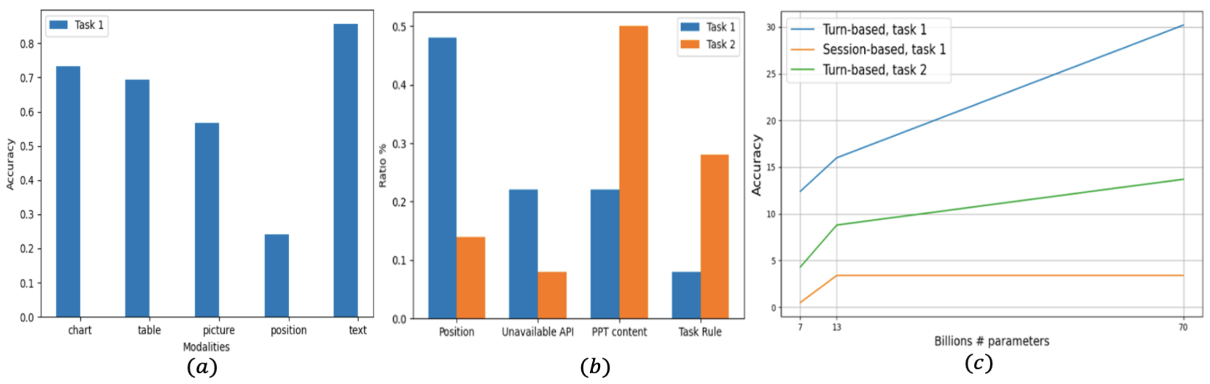


Figure 4: We illustrate the analysis results of the creating new PPT file task (task 1) and the editing PPT template task (task 2). In sub-figure (a), we report the average turn-based accuracy for instructions involving chart, table, picture, position, and pure text. We don't draw the accuracy of task 2 as no chart instruction in this task. In sub-figure (b), we report the ratio of four common errors made by GPT-4. In sub-figure (c), we report the accuracy with the model size. We don't plot the session-based accuracy of task 2 as it is zero.

for GPT-4 and GPT-4 exhibits its weakest performance in instructions involving position-related operations, with only 24% accuracy. This underscores GPT-4's limitations in spatial perception ability.

5 Analysis

In this section, we analyze the reasons for GPT-4's errors. We further analyze the influence of model size and dialogue history.

5.1 *Error Analysis of GPT-4 in our benchmark*

To analyze the error made by GPT-4, in our benchmark, we gather 50 wrong samples for each of the two tasks in our benchmark in the turn-based evaluation. We find that these wrong samples fall into four error types and visualize the distribution of these four main error types in Figure 4 (b): (1) Position errors: These occur when GPT-4 struggles with instructions involving position adjustments. For example, when asked to move the shape to the bottom of the slide, GPT-4 wrongly calls the "set_top" API. (2) Calling unavailable APIs: GPT-4 sometimes generates APIs that don't actually exist in the reference API file, resulting in what we call the "API hallucination problem." (3) Misunderstanding PPT file content: GPT-4's wrong comprehension of the PPT content leads to incorrect APIs. For example, when instructed to make the font size of the current slide's title consistent with previous slides, GPT-4 set a font size that is different from what was used in previous slides' titles. Misunderstanding the PPT content becomes the main error in the editing template task. (4) Unfollowing Powerpoint task rules: LLMs don't understand the Powerpoint task rules. For instance, GPT-4 may directly insert the new content into the original title when instructed to rewrite the title. For the session-based evaluation, we also collect 50 wrong examples. And we find that the main reasons for errors are similar. One unique phenomenon in this evaluation is that the LLM would repeat previous errors (e.g., repeatedly employing infeasible APIs) in subsequent turns.

We analyze the error of open-source LLMs and explain the reasons for their performance gap compared to closed-source LLMs in Appendix H.

5.2 *Does bigger LLM work better on PPTC?*

To investigate how the model size impacts the LLM's performance in our benchmark, we conduct tests using LLaMa-2-chat LLM with 7, 13, and 70 billion parameters and plot the results in Figure 4 (c). We observe that larger LLM consistently achieve higher turn-based accuracy for both the creating new PPT and editing PPT template tasks. For example, in the creating new PPT file task, we find that the turn-based accuracy increases from 13.2 (7B) to 30.1 (70B). However, we do not observe a clear positive correlation between model size and session-based performance. One possible explanation is that although the 70B LLM can correctly finish more intermediate steps, it still falls short of completing the entire session. A larger LLM may be necessary.

5.3 *Does dialogue history help LLMs to generate the API sequence?*

To investigate the influence of dialogue history in our prompt (see Figure **??**), we make an ablation experiment for the dialogue history component of our turn-based evaluation prompt[3]. In this evaluation, the dialogue history contains previous turns along with their feasible API sequences. When we removed the dialogue history from the prompt, we observed a decline in GPT-4's performance. Specifically, GPT-4 drops its performance from 75.1 % to 73.1 % in the creating new PPT file task and decreases its performance by 6.2 % in the editing template task. This experiment shows the positive effect of the dialogue history, as it helps the LLM to both understand the dialogue background and instruct the LLM to correctly use the APIs, similar to few-shot demonstration examples.

6 Related Works

Large Language Models like ChatGPT, GPT-4 (Bubeck et al., 2023; OpenAI, 2023), and Bard have billions of parameters and have been trained on the Internet corpus with trillions of tokens. They can write code (Liu et al., 2023a), prove mathematical theorems (Jiang et al., 2022), pass the professional exam (Zhong et al., 2023; Gilson et al., 2023; Katz et al., 2023), employ other models and APIs (Schick et al., 2023; Liang et al., 2023; Wu et al., 2023; Patil et al., 2023), and also perform well on other basic natural language tasks (Kim et al., 2023; Jiao et al., 2023; Zhong et al., 2023; Wang et al., 2023). That raises the hope of achieving artificial general intelligence (AGI).

To further boost LLM's performance on the specific task, one approach involves prompting engineerings, such as the chain of thought prompting (Wei et al., 2022; Shi et al., 2022; Yao et al., 2023), self-consistency (Wang et al., 2022) and the least to most prompting (Zhou et al., 2022). Another approach aims to use feedback to improve performance. The self-refine method (Madaan et al., 2023; Shinn et al., 2023) refines the output through iterative feedback and refinement Provided by LLM itself or sparse reward signal. The learning to program method (Guo et al., 2023) learns the task program by inducing the general solutions from the errors (feedback) iteratively and uses the program to guide the test inference.

Task completion benchmarks for measuring large language models. To measure LLM's task completion performance, Saycan (Brohan et al., 2023) and VirtualHome (Puig et al., 2018) benchmarks ask LLM to generate the correct action sequence for controlling the robot to finish user instruction. WebShop (Yao et al., 2022) and Android in the wild (Rawles et al., 2023) ask LLM to navigate websites and conduct actions to meet the user requirement. APIBench (Patil et al., 2023) and ToolBench (Xu et al., 2023b; Qin et al., 2023b) involve selecting and using APIs to complete the task instruction. Agentbench (Liu et al., 2023b)assesses LLM as autonomous agents in 8 environments and WebArena (Zhou et al., 2023) considers task completion in web-based interactions. We make a comprehensive comparison for these benchmarks in Appendix I.

7 Conclusion

We introduce the PowerPoint Task Completion benchmark to measure LLMs' ability to complete multi-turn user instructions within the context of the multi-modal PowerPoint software. We further propose the PPTX-evaluation system to access and compare the performance of 10 LLMs and 2 LVMs. We further analyze the behavior of LLMs and find three main error factors that limit their performance. Our benchmark and findings can help the research community design better AI assistants.

8 Limitations and potential risks

Our benchmark does not consider instructions that involve subjective evaluation. For example, the user may want to make the slide more beautiful. However, it's hard to automatically evaluate if the generated file (the model output) is more beautiful. Another limitation is that we do not consider the instructions that need non-API operations. For example, the user may want to draw a cat on the slide. That instruction needs the AI-assistant system to draw the cat by dragging the mouse and is still infeasible for LLMs and LLM-based systems. We only consider instructions that can be completed by directly executing the API sequence.

For privacy concerns, we replace all real human names, emails, and addresses with those generated randomly. Completing the virtual PPT task could not hurt humans and we do not see any further potential risk in our benchmark.

[3]The task instruction, current user instruction, API file, PPT content in the prompt are necessary parts for generating the API sequence. So we don't conduct ablation studies on them.

References

Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian

Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. 2023. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*, pages 287–318. PMLR.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*.

Aidan Gilson, Conrad W Safranek, Thomas Huang, Vimig Socrates, Ling Chi, Richard Andrew Taylor, David Chartash, et al. 2023. How does chatgpt perform on the united states medical licensing examination? the implications of large language models for medical education and knowledge assessment. *JMIR Medical Education*, 9(1):e45312.

Yiduo Guo, Yaobo Liang, Chenfei Wu, Wenshan Wu, Dongyan Zhao, and Nan Duan. 2023. Learning to program with natural language. *arXiv preprint arXiv:2304.10464*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Albert Q Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. 2022. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. *arXiv preprint arXiv:2210.12283*.

Wenxiang Jiao, Wenxuan Wang, Jen-tse Huang, Xing Wang, and Zhaopeng Tu. 2023. Is chatgpt a good translator? a preliminary study. *arXiv preprint arXiv:2301.08745*.

Daniel Martin Katz, Michael James Bommarito, Shang Gao, and Pablo Arredondo. 2023. Gpt-4 passes the bar exam. *Available at SSRN 4389233*.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2023. Language models can solve computer tasks. *arXiv preprint arXiv:2303.17491*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Minghao Li, Feifan Song, Bowen Yu, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Apibank: A benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*.

Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. 2023. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *arXiv preprint arXiv:2303.16434*.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023a. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *arXiv preprint arXiv:2305.01210*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2023b. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*.

OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. 2018. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8494–8502.

Chengwei Qin, Aston Zhang, Zhuosheng Zhang, Jiaao Chen, Michihiro Yasunaga, and Diyi Yang. 2023a. Is chatgpt a general-purpose natural language processing task solver? *arXiv preprint arXiv:2302.06476*.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, Bokai Xu, Zhen Zhang, Yining Ye, Bowen Li, Ziwei Tang, Jing Yi, Yuzhang Zhu, Zhenning Dai, Lan Yan, Xin Cong, Yaxi Lu, Weilin Zhao, Yuxiang Huang, Junxi Yan, Xu Han, Xian Sun, Dahai Li, Jason Phang, Cheng Yang, Tongshuang Wu, Heng Ji, Zhiyuan Liu, and Maosong Sun. 2023b. Tool learning with foundation models.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023c. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2023. Android in the wild: A large-scale dataset for android device control. *arXiv preprint arXiv:2307.10088*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, et al. 2022. Language models are multilingual chain-of-thought reasoners. *arXiv preprint arXiv:2210.03057*.

Noah Shinn, Beck Labash, and Ashwin Gopinath. 2023. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Zengzhi Wang, Qiming Xie, Zixiang Ding, Yi Feng, and Rui Xia. 2023. Is chatgpt a good sentiment analyzer? a preliminary study. *arXiv preprint arXiv:2304.04339*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. 2023. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023a. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. 2023b. On the tool manipulation capability of open-source large language models. *arXiv preprint arXiv:2305.16504*.

Aiyuan Yang, Bin Xiao, Bingning Wang, Borong Zhang, Chao Yin, Chenxu Lv, Da Pan, Dian Wang, Dong Yan, Fan Yang, et al. 2023. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2023. Agieval: A human-centric benchmark for evaluating foundation models. *arXiv preprint arXiv:2304.06364*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Chi. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

A The API Reference File, PPT reader, and API excutor

We list all APIs and their descriptions in Figures 7 and 8. We provide 49 feasible APIs.

Compared to the general AI assistant, LLMs still have two limitations for completing the task in our benchmarks: (1) LLMs can not directly process the PPT file. So we provide a PPT reader function that extracts all shapes and their information from the PPT file and transforms them into the text format as the PPT file content. Then LLMs can understand and process the PPT file content. The code of the PPT reader is in the supplementary. We illustrate two examples for how the PPT reader turns the shapes (e.g., title, figure and table) of the PPT file into the text description of them in Figure 5 & 6. (2) LLMs cannot directly use PPT software through a keyboard and mouse. Therefore, we have defined

PPT APIs based on the operational logic within the PPT software. and provide an implementation for these APIs in Python that can swiftly generate PPT files. In future work, it may be possible to explore the use of large multimodal models to understand on-screen content and implement APIs using a keyboard and mouse.

B Details for the dataset collection and validation

In the collection process, we employ 6 skilled crowd workers. Our crowd workers comprise professional data science engineers well-versed in PowerPoint. Each worker takes on a specific role in the instructional writing work and is encouraged to write instructions in his/her own words. Workers were asked to spend at least 20 minutes on every session. We delete repeated sessions and short sessions that have no more than 50 tokens. For the editing PPT template task, the 50 PPT templates are collected from the SlidesCarnival website (https://www.slidescarnival.com/). SlidesCarnival is a free and open-source PPT template website. Each session in the editing task has a unique template.

To ensure the data quality of this benchmark, the principal engineer reviews and refines the instructions and API sequences written by the above 7 workers for initial quality assurance. Then the three authors of this paper further undertake the following validation steps: (1) Assessing Instruction Clarity and Relevance: They examine whether the instructions are clear, contextually related to the session topic, and align with the ongoing conversation. (2) API Sequence Execution: The authors execute the provided API sequences to identify and rectify coding errors. (3) Goal Achievement Check: They verify if the instructions' intended goals are successfully completed in the label files.

In the event that errors are identified during this validation process, the authors promptly report them to the respective workers for revision. The three authors are computer science senior students and researchers.

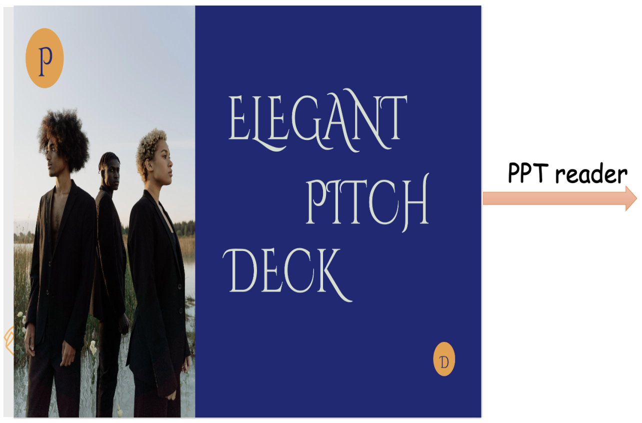
C The Details for PPTX-Match Evaluation System

Specifically, our PPTX-Match Evaluation System first uses a Python-PPTX Content Reader Module to iterate over all shapes in the prediction PPT file produced with the LLM and the label output file. A shape in the PPTX library typically refers to an individual object, such as a text box or table. Then our system extracts attributes like text, style, and position of the shapes using the PPTX library. Next, we check all attributes from the prediction PPT file. For non-position attributes (e.g., text content), we first convert it and the corresponding attribute in the label PPT file into two strings, and then we use the Exact Match method to examine if the two strings are the same. If they are different or we do not find the corresponding attribute in the label file, then we find an incorrect match. For the position attribute (e.g., location information), we focus on checking if the objects in the prediction PPT file satisfy the correct position relation <A, B, REL>, where A and B are objects that should satisfy the relation REL. In the benchmark collection process, we ask crowd workers to label the position relation that objects should satisfy to finish the instruction.

If there are no incorrect matches for all non-position attributes and no rule violations for all position-related attributes, we consider the LLM has successfully completed the user instruction.

D The Prompt for Content Selection Algorithm

We put the prompt of content selection algorithm in Figure 9.



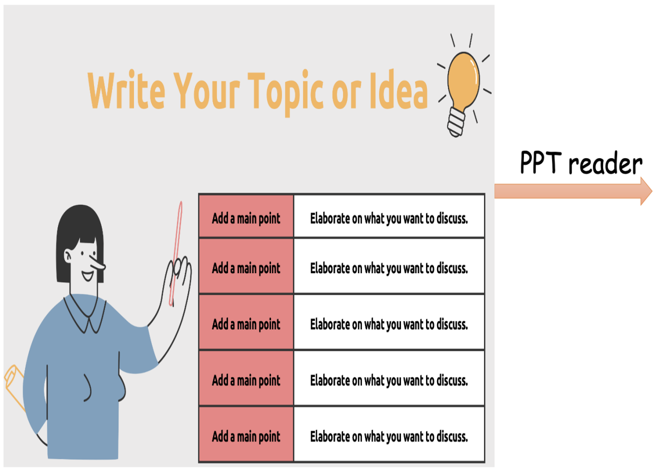
```
There are 29 slides with slide height 5143 and slide width 9144.
Slide 0 with background color D4DECF:
[Picture 0]
Size: height=5143, width=3561
Picture Style: rotation=0
Visual Positions: left=0, top=0


[TextBox 0]
Size: height=914, width=4164
Text: ELEGANT
Font Style: bold=False, italic=False, underline=False, size=762000,
color=D4DECF, fill=None, font style=Cinzel Decorative, line_space=1.2,
align=LEFT (1)
Visual Positions: left=4172, top=855


[TextBox 1]
Size: height=923, width=3198
Text: PITCH
Font Style: bold=False, italic=False, underline=False, size=762000,
color=D4DECF, fill=None, font style=Cinzel Decorative, line_space=1.2,
align=LEFT (1)
Visual Positions: left=5676, top=1903


[TextBox 2]
Size: height=914, width=2463
Text: DECK
Font Style: bold=False, italic=False, underline=False, size=762000,
color=D4DECF, fill=None, font style=Cinzel Decorative, line_space=1.2,
align=LEFT (1)
Visual Positions: left=4172, top=2756
```

Figure 5: The PPT reader transforms the figure and title into their text form.


Write Your Topic or Idea
Add a main point
Elaborate on what you want to discuss.
Add a main point
Elaborate on what you want to discuss.
Add a main point
Elaborate on what you want to discuss.
Add a main point
Elaborate on what you want to discuss.
Add a main point
Elaborate on what you want to discuss.
PPT reader

```
Slide 7 with background color EBEAEA:
[Picture 0]
Size: height=5479, width=6935
Picture Style: rotation=0
Visual Positions: left=-99, top=4202


[TextBox 0]
Size: height=1278, width=13155
Text: Write Your Topic or Idea
Font Style: bold=True, italic=False, underline=False, size=1054354,
color=000000, fill=None, font style=Ubuntu, line_space=1.11997,
align=CENTER
(2) Visual Positions: left=2566, top=1271


[Table] with 5 rows and 2 columns
Size: height=5775, width=10246
Data:
|Add a main point|Elaborate on what you want to discuss.|
|Add a main point|Elaborate on what you want to discuss.|
|Add a main point|Elaborate on what you want to discuss.|
|Add a main point|Elaborate on what you want to discuss.|
|Add a main point|Elaborate on what you want to discuss.|
Visual Positions: left=7012, top=3850


[Picture 1] Size: height=2234, width=2048
Picture Style: rotation=17
Visual Positions: left=15462, top=743
```

Figure 6: The PPT reader transforms the figure, table and title into their text form.

API reference file

Slide-related APIs

API: create slide(): This API creates a new slide.
API: move to previous slide(): This API moves to the previous slide.
API: move to next slide(): This API moves to the next slide.
API: move to slide(slide id): This API moves to the slide with given slide id.It takes one parameter 'slide id', the ID of the slide to move to as a integer.

Choose-related APIs

API: choose title(): This API selects the title on the slide. You should first call choose title() before inserting text to or changing font attributes of the title.
API: choose content(): This API select the content on the slide. You should first call choose content() before inserting text to or changing font attributes of the content.
API: choose textbox(idx): This API selects the textbox element on the slide. It takes one parameter, the index of textbox as integer. idx is set to 0 by default, meaning the first textbox. You should first call choose textbox() before inserting text to or changing font attributes of the textbox element.
API: choose picture(idx): This API selects the picture element on the slide. It takes one parameter, the index of textbox as integer. idx is set to 0 by default, meaning the first textbox. You should first call choose picture() before changing height, width, rotation of the picture element. You should not call choose picture() before inserting picture element.
API: choose chart(): This API selects the chart element on the slide. You should first call choose chart() before changing the chart. You should not call choose chart() before inserting chart element.
API: choose shape(shape name): This API selects a specific shape by shape name on the slide. It takes one parameter 'shape name', the name of the shape to select as a string. shape name can be chosen from ['rectangle','right arrow','rounded rectangle','triangle','callout','cloud','star','circle'] You should first call choose shape(shape name) before you can do operations on the shape. You should not call choose shape(shape name) before inserting shape element.
API: choose table(): This API selects the table element on the slide. You should first call choose table() before changing the table. You should not call choose table() before inserting table element.
API: choose table cell(row id, column id): This API selects a specific cell in the table by giving row id and column id. It takes two parameters, the row id and column id of the cell to select as integers (id starts from 0). Remember the first parameter is row id, the second parameter is column id. You should first call choose table cell(row id, column id) before inserting text.

Basic APIs

API: set background color(color): This API sets the background color of the slide. It takes one parameter 'color', the color name to set as a string, such as 'red', 'purple'.
API: set width(width): This API sets the width of the selected object. It takes one parameter 'width', the width of an object in centimeters as float. You should first choose an object before you can change the width of it.
API: set height(height): This API sets the height of the selected object. It takes one parameter 'height', the height of an object in centimeters as float. You should first choose an object before you can change the height of it
API: rotate element(angle): This API rotates the selected element by the specified angle. It takes one parameter 'angle', the angle to rotate clockwise as integer. You should first choose an object before you can rotate it.
API: set fill color(color): This API sets the fill color of the selected object after the object is chosen. It takes one parameter 'color', the color name to set as a string, such as 'red', 'purple'. You can set the fill color of content, title or textbox.
API: set left(left): This API moves and changes the object's position. It sets the x position of the selected object's leftmost point. It takes one parameter, the x position to set. You should first choose an object before you can change the left of it
API: set top(top): This API moves and changes the object's position. It sets the y position of the selected object's upmost point. It takes one parameter, the y position to set. You should first choose an object before you can change the top of it.

Figure 7: The reference API file: part 1.

API reference file

Text-related APIs
API: insert text(text): This API inserts text into a text frame (textbox, title, content, table).
API: insert bullet point(text): This API inserts a bullet point into the content. It takes one parameter, the text of the bullet point to insert as a string.
API: insert note(text): This API inserts a note onto the slide. It takes one parameter, the note text to insert as a string.
API: insert textbox(): This API inserts a textbox onto the slide. When you need to add a caption or text under/above/left to/right to an object, you can call insert textbox().
API: delete text(): This API delete the text part of an object. You should first choose content or title before you can call delete text()
API: set font size(font size): This API sets the size of the font It can take one argument 'font size', the font size to set as an integer.
API: set font color(color): This API sets the color of the font. It takes one parameter 'color', the color name to set as a string, such as 'red', 'purple'.
API: set font bold(): This API sets the font to be bold.
API: set font italic(): This API sets the font to be italic.
API: set font underline(): This API sets the font to be underlined.
API: set font style(font name): This API sets the font style of the selected text. It can take one argument 'font style', the font name as a string.
API: set line space(line space level): This API sets the line spacing of the selected text. It can take one argument 'line space level', as an integer, default 0.
API: text align left(): This API aligns the text to left.
API: text align center(): This API aligns the text to center.
API: text align right(): This API aligns the text to right.
Image and shape-related APIs
API: insert picture(picture name): This API inserts a picture onto the slide. It takes one parameter 'picture name', the name or description of picture as a string
API: insert rectangle(): This API inserts a rectangle or square shape onto the slide.
API: insert right arrow(): This API inserts an arrow shape onto the slide.
API: insert rounded rectangle(): This API inserts a rounded rectangle shape onto the slide.
API: insert triangle(): This API inserts a triangle shape onto the slide.
API: insert callout(): This API inserts a callout shape onto the slide.
API: insert cloud(): This API inserts a cloud shape onto the slide.
API: insert star(): This API inserts a star shape onto the current slide.
API: insert circle(): This API inserts a circle or oval shape into the current slide.
Table-related APIs
API: insert table(row num, col num): This API inserts a table of row num rows and col num columns onto the current slide. It takes two argument, the row number and the column number of the inserted table as integer. Remember the first parameter is row number and the second parameter is column number.
API: insert table row(row data): This API inserts a row (list) of data into the table. It takes one argument, the data to insert as a list of numbers or strings. You should first call choose table() before you can call insert table row(). The parameter 'row data' should be a list of strings.
Chart-related APIs
API: insert line chart(data, series): This API inserts a line chart onto the slide. It takes two argument, 'data' is a list of numbers and 'series' is a list of strings.
API: insert bar chart(data, series): This API inserts a bar chart onto the slide. It takes two argument, 'data' is a list of numbers and 'series' is a list of strings.
API: insert pie chart(data, series): This API inserts a pie chart onto the slide. It takes two argument, 'data' is a list of numbers and 'series' is a list of strings.
API: set chart title(title): This API sets the title of a previously inserted chart. It takes one argument 'title', the title to be set as a string.

Figure 8: The reference API file: part 2.

Content Selection prompt

You are an AI assistant for PowerPoint. Your task is to determine what kind of content is necessary to fulfill the user's instruction. You have an API to extract the content, please call the get_content api with correct parameters to fulfill the user's instruction. You need to extract the minimum necessary information to fulfill user's instruction.

Get_content API: get_content(need_text: Indicates whether text information is required. The text information encompasses text in title, content, textbox, table, chart, and shape. This parameter is particularly useful when inserting or modifying text of title, content, textbox, table, chart, and shape, or when information about these objects is essential.
need_style: Indicates whether style information is required. Style information includes attributes like font type, font size, color, background color, line space, bold, undeline, italic and other visual aspects of objects like rotation. This is useful when changing the appearance of text or objects or when information about an object's appearance is essential.
need_position: Indicates whether position information is required. The position details encompass an object's height, width, and its left and top positions. This is crucial when moving objects or altering an object's size.
need_title: Determines if information related to the title is required.
need_content: Determines if information related to the content is required.
need_picture: Determines if information related to the picture is required.
need_table: Determines if information related to the table is required.
need_chart: Determines if information related to the chart is required.
need_textbox: Determines if information related to the textbox is required.
need_shape: Determines if information related to the shapes (rectangle, right arrow, rounded rectangle, triangle, callout, cloud, star, circle) is required.)
Where the parameters are either 1 (needed) or 0 (not needed). You should only answer with calling get_content() with the right parameters.

For examples:
Instruction: Increase the font size of the content to 20.
Explanation: For information, style information (font size) is needed. For objects, content is needed.
Answer:
get_content(need_text=1,need_style=1,need_position=0,
need_title=0,need_content=1,need_picture=0,need_
table=0,need_chart=0,need_textbox=0,need_shape=0)
...
Given the instruction, output the Answer without Explanation:
Instruction: <Current user instruction>
Answer:

Figure 9: The prompt of the content selection algorithm.

E Introductions of 10 LLMs and 2 LVMs

We consider the following 10 LLMs:

- ***GPT-4*** (OpenAI, 2023): The latest LLM in the GPT series. GPT-4 is a cutting-edge, large-scale generative pre-trained transformer model. It offers improved performance and a wider knowledge base compared to its predecessors. It showcases human-level proficiency in several scenarios.
- ***ChatGPT***: ChatGPT is a conversational AI model crafted for dynamic interactions. It's learned from extensive instruction data and fine-tuned through reinforcement learning with human feedback (RLHF). This empowers it to deliver responses that align with human expectations, maintaining context and coherence in conversations.
- ***Text-Davinci-003*** (Brown et al., 2020): GPT-3.5 sits between GPT-3 and GPT-4, enhancing performance via additional instruction tuning. It acts as a link between these models, facilitating comparison. We've chosen the **Text-Davinci-003** variant from the GPT-3.5 series for our evaluation.
- ***LLaMa-2-Chat*** (Touvron et al., 2023): LLaMa 2, an auto-regressive open-source language model, employs an optimized transformer design. Chat versions utilize supervised fine-tuning (SFT) and reinforcement learning with human feedback (RLHF) to match human preferences for helpfulness and safety.
- ***Baichuan-Chat***: It is a transformer model trained on approximately 1.2 trillion tokens. It supports both Chinese and English, with a context window length of 4096.
- ***Baichuan-2-Chat*** (Yang et al., 2023): It is a large-scale multilingual language model trained from scratch, on 2.6 trillion tokens. The chat version uses Supervised Fine-Tuning (SFT) and Reinforcement Learning from Human Feedback (RLHF) to align with humans.
- ***WizardLM v1.2*** (Xu et al., 2023a): WizardLM v1.2 is finetuned from LLaMa 2 using supervised instruction fine-tuning, where instructions are created by rewriting the initial instructions step by step.
- ***Vicuna v1.5 (16k)*** (Chiang et al., 2023): Vicuna v1.5 (16k) is finetuned from LLaMa 2 using supervised instruction fine-tuning and linear RoPE scaling. It's trained on about 125K conversations sourced from ShareGPT.com.
- ***Code-LLaMa-instruct*** (Chiang et al., 2023): Code-LLaMa is a LLaMa-based LLM designed for general code completion and understanding. Its instruction version further supports the chat function with users. Code LLaMa models feature a multitask training objective consisting of both autoregressive and causal infilling prediction (predicting the missing part of a program given a surrounding context).
- ***Mistral-7b-instruct*** (Jiang et al., 2023) Mistral-7b is a 7 billion size LLM that leverages grouped-query attention (GQA) for faster inference, coupled with sliding window attention (SWA) to effectively handle sequences of arbitrary length. It outperforms many larger LLMs (e.g., LLaMa-13b) on public evaluation benchmarks. We use its instruction version.

F The details and inference prompts for turn-based and session-based evaluations

In the turn-based evaluation, we assume that the previous turns in one session have been correctly finished. We prompt the LLM to generate the API sequence to finish the current turn's user instruction. The prompt consists of the task instruction for finishing the current user instruction, the API file containing feasible APIs, the parsed PPT file content from the PPT file, and dialogue history consisting of instructions of previous turns with their feasible API sequences. In the session-based evaluation, we prompt the LLM to finish all turns in a session sequentially. The prompt in this evaluation has two differences: the API solutions for previous turns in dialogue history are the outputs of the LLM instead of the correct API sequences. (2) The PPT content is parsed from the PPT file obtained by executing the previous outputs of the LLM. That means the error made by LLMs in previous turns would influence subsequent turns. We list the inference prompt we used in Figure **??**.

G Experimental details

Azure OpenAI services[4] offer two API types: completion and chat completion. Completion API generates text from prompts, while chat completion API responds based on conversation history and new input. We use the completion API for Text-Davinci-003 and the chat completion API for ChatGPT and GPT-4. We set a temperature of zero for deterministic output and a max token limit of 2048. The frequency penalty and top p are kept at their default values of zero and 1, respectively. We use the text-embedding-ada-002 API as the embedding API in the API selection algorithm and set k as 15. If the token number of the input prompt is beyond the token limit, we cut the PPT file content to reduce the token number of the prompt. For the ToT method, we follow the official code to run it[5].

H Error Analysis for open-source LLMs

To analyze the errors of open-source LLMs and explain why their performance is lower than that of the closed-source LLM (e.g. GPT-4), we collect 50 wrong examples made by WizardLM and CodeLLaMa in turn-based evaluation, respectively. Each example consists of a user instruction and a wrong prediction file compared to the label file.

Drawing from incorrect examples, we find three unique error reasons in open-source LLMs to explain their poor performance :

- ***Unexcutable API prediction sequence*** We find that open-source LLMs frequently generate API sequences that are either meaningless or improperly formatted. For instance, the WizardLM exhibits a 37% rate of generating incorrect API sequences such as 'API(), API()' in its erroneous examples. Although CodeLLaMa is less prone to this type of error, it still generates API sequences like 'set_font_size(15, 2, 5)' that would return errors 'ERROR: set_font_size() takes 1 positional argument but 3 were given', with a rate of 25%. In contrast, closed-source LLMs like GPT-4 rarely make these mistakes, showcasing excellent code generation abilities and the capability to generate concise and clear API sequences.

- ***Severe API hallucinations*** We find that both two open-source LLMs often call APIs that do not present in our API reference file, such as 'delete_table()', 'choose_slide' (with the correct API being 'move_to_slide'), and 'choose_slide_by_id'. These mistakes constitute half of the errors made by these LLMs. While GPT-4 sometimes also calls unavailable APIs, it does so at a lower rate.

- ***Short context size limits the input of the PPT file*** Open-source LLMs typically have a limited context size of 2k tokens, which proves insufficient for handling lengthy PowerPoint (PPT) template tasks, often comprising multiple slides. Our analysis reveals that the input for open-source LLMs generally includes file information from only the initial 1 to 3 slides. Consequently, these models face challenges in executing instructions related to subsequent slides and cross-slide operations. This limitation leads to the generation of APIs containing incorrect file information or invoking improper APIs. In contrast, closed-source LLMs such as GPT-4 support an extended context size of 32k tokens, enabling them to effectively manage longer templates.

I A Comparison of Task Completion Benchmarks

In Table 3, we compare our benchmark with other task completion benchmarks in five aspects. Our

[4]https://azure.microsoft.com/en-us/products/cognitive-services/openai-service

[5]ToT:https://github.com/princeton-nlp/tree-of-thought-llm

benchmark is the only one that satisfies the five requirements.

Resource	PPTC Our work	Saycan (Brohan et al., 2023)	VirtualHome (Puig et al., 2018)	WebShop (Yao et al., 2022)	ToolBench (Qin et al., 2023c)	APIBench (Patil et al., 2023)	API-Bank (Li et al., 2023)	ToolAlpaca (Tang et al., 2023)	AgentBench (Liu et al., 2023b)	WebArena (Zhou et al., 2023)
Real-world API?	✔	✔	✔	✔	✔	✖	✔	✖	✔	✔
Multi-tools call?	✔	✔	✔	✔	✔	✖	✔	✖	✔	✔
Multi-turn interaction?	✔	✖	✖	✖	✖	✖	✖	✖	✖	✖
Multi-modal environment?	✔	✔	✔	✖	✖	✖	✖	✖	✔	✔
Evaluation based on final status?	✔	✖	✖	✔	✖	✖	✖	✖	✔	✔

Table 3: A comparison of our PPTC benchmark for task completion