m&m's: A Benchmark to Evaluate Tool-Use for *m*ulti-step *m*ulti-modal Tasks

Zixian Ma[1], Weikai Huang[1], Jieyu Zhang[1], Tanmay Gupta[2], Ranjay Krishna[1,2]

[1] University of Washington
[2] Allen Institute of Artificial Intelligence

A Additional data

Table 1: We list all 33 tools across three categories - ML models, public APIs, and image processing modules - in *m&m*'s.

Tool category	Tool name
ML model	text generation, text summarization, text classification, question answering, optical character recognition, image generation, image editing, image captioning, image classification, image segmentation, object detection, visual question answering, automatic speech recognition
Public APIs	get weather, get location, get math fact, get trivia fact, get year fact, get date fact, search movie, love calculator, wikipedia simple search
Image processing	image crop, image crop top, image crop bottom, image crop left, image crop right, select object, count, tag, color pop, emoji, background blur

We present more examples of query-plan pairs of *m&m*'s in Figure 1, and a complete list of all 33 tools in Table 1. For more details about the tools and their implementation, please refer to our Github codebase[3].

B Dataset generation

B.1 Tool graph

We include a visualization of the full tool graph used in our dataset generation pipeline (Figure 2).

B.2 Prompts

We generate the queries with the prompt in Figure 3, and rewrite the argument values of `text generation` and `image generation` with the prompt shown in Figure 4.

[3] https://github.com/RAIVNLab/mnms

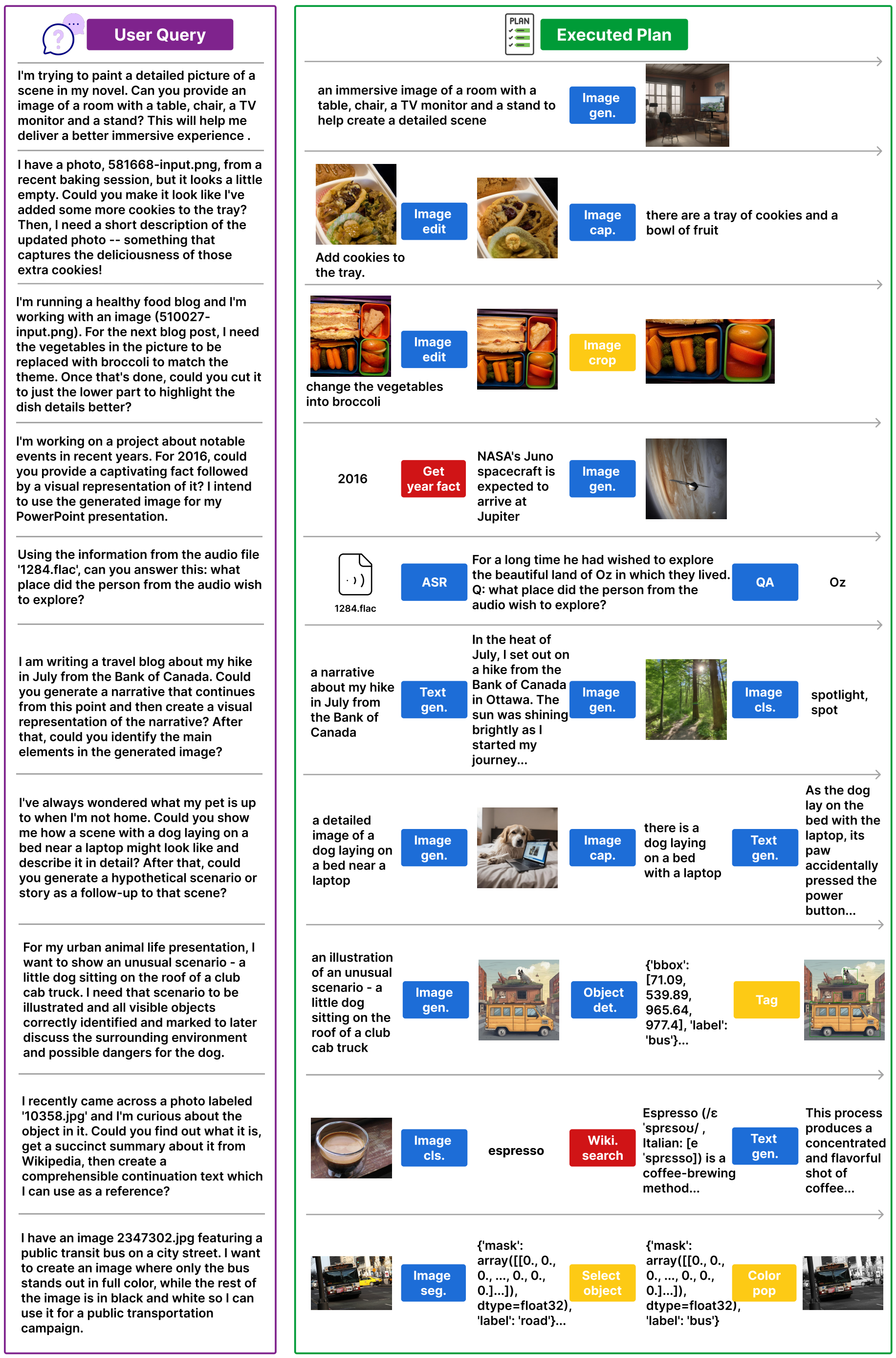

Fig. 1: We present additional examples of query-plan pairs along with the execution results of the plans in $m\&m$'s.

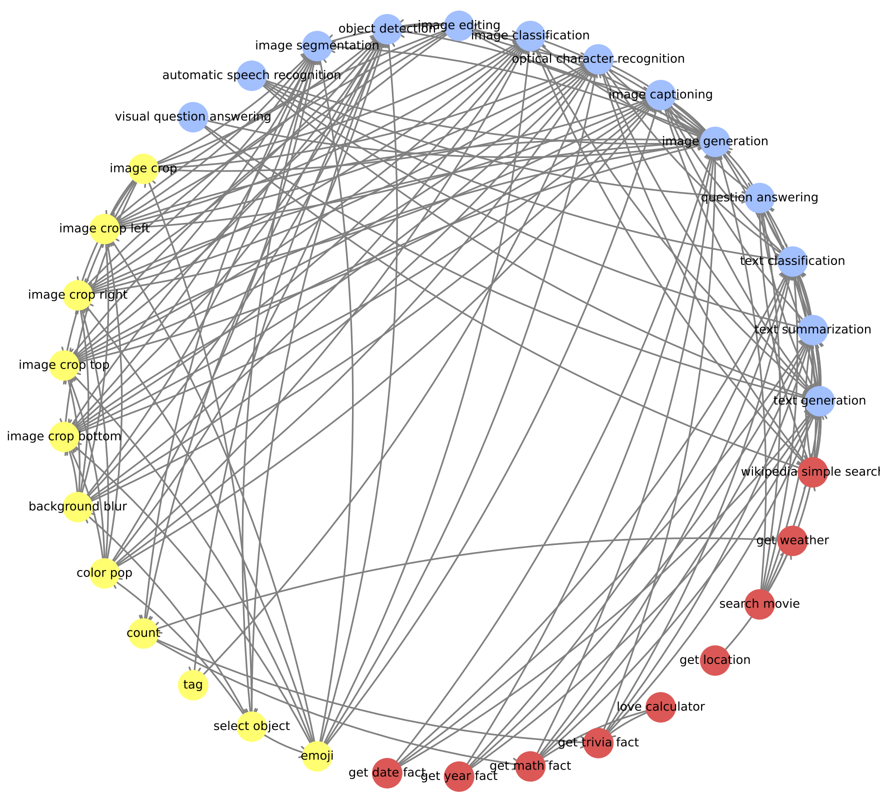

Fig. 2: The full tool graph consists of 33 unique tools as nodes (red = public APIs, yellow = image processing tools, blue = machine learning models) and valid connections between them as edges.

I have these tools:
image classification: It takes an image and classifies the subject in the image into a category such as cat or dog.
wikipedia simple search: Perform a basic search query on Wikipedia to retrieve a summary of the most relevant page.

Can you write 2 example queries for tasks I can do with a combined workflow of image classification, followed by wikipedia simple search?

There are a few requirements:
1) Each task query should sound natural, represent a realistic use case, and should NOT mention image classification, wikipedia simple search.
2) Each query should be based on these inputs to image classification: {'image': '16611.jpg'} and should explicitly mention these inputs.

Fig. 3: Query generation prompt. We present the full prompt used for query generation.

B.3 Human verification statistics

The pairwise agreement rates among the 3 annotators are 74.95%, 81.43%, 70.88%, and the average pairwise agreement rate is 75.75% (std=4.34%).

B.4 Data filtering

We perform two types of data filtering on the 1565 human-verified examples: (1) we manually filter out 349 examples with poor execution results, especially those where intermediate tools return wrong or empty outputs (e.g. when `question answering` is the second tool in the sequence and outputs an empty string); (2) we filter out a total of 334 examples whose plans involve `image generation` and have more than 4 unique queries. We perform the second filtering step because of two reasons. First, the frequency of the tools initially follows the distribution in Figure 5 (blue), where `image generation` has a much higher count – 918 – than other tools. Thus, we would like to reduce the frequency of `image generation` in the dataset while maintaining the frequency of rare tools. To achieve this while also preserving the diversity of tool plans, we choose to filter out examples whose plans have 5-10 unique queries, as the average number of unique requests per tool plan before filtering is 4.20. We end up filtering out 40% (or 349) of these examples. After these two filtering steps, we are left with 882 examples in total that follow the distribution in Figure 5 (red).

B.5 Alternative plans

In addition to the one human verified groundtruth plan, we have also generated alternative plans to supplement our evaluation. Concretely, we generate these alternative plans in three steps: first, we generate a set of syntactically valid (i.e. the alternative tool's input and output types are correct) and semantically valid (i.e. the alternative tool performs the same functionality as the original tool) alternative tools for each tool in our toolset; second, we manually verify their validity and only keep the human-verified valid tools in the alternative tools set; finally, we compose all valid tools at each position in the plan to obtain all combinations as the total set of valid plans. To generate the syntactically valid tools, we create a graph with both data (including input and output) and tools as nodes, and we obtain the syntactic alternative tools t_o^{alt} of the original tool t_o

```
# INSTRUCTION #:
A tool node is defined as a dictionary with keys “id” storing its unique identifier, “name” specifying the model to call, and “args” specifying
the arguments needed to make an inference call to this tool.
Your task is to rewrite ONLY the 'text' values in the tool nodes 'text generation' and 'image generation' based on the user request so that
they are more concrete and aligned with user's intentions.

Below are a few examples:
# EXAMPLES #:
Request: I'm creating an educational video about the world's fastest vehicles and I need material on watercrafts. Could you provide me with
a thorough explanation and some engaging facts on What's The Fastest Boat Ever Made?
Nodes: [{'id': 0, 'name': 'text generation', 'args': {'text': "What's The Fastest Boat Ever Made?"}}]
New nodes: [{'id': 0, 'name': 'text generation', 'args': {'text': " a thorough explanation and some engaging facts on "What's The Fastest Boat
Ever Made?"}}]

Request: I would like to create a dynamic visual for my blog post about baseball. The text description I have is 'There is a baseball player
who swung for the ball'. Could we use that to come up with something eye-catching and fitting for the topic?
Nodes: [{'id': 0, 'name': 'image generation', 'args': {'text': 'There is a baseball player who swung for the ball'}}]
New nodes: [{'id': 0, 'name': 'image generation', 'args': {'text': 'a dynamic and eye-catching image of a baseball player who swung for the
ball'}}]

Request: For a blog topic heading 'What Really Happens When You Flush on an Airplane?', I'm trying to explain the process visually to my
readers. Could you first generate a comprehensive, easy-to-understand description of the process, and then create an illustrative image
based on that description?
Nodes: [{'id': 0, 'name': 'text generation', 'args': {'text': 'What Really Happens When You Flush on an Airplane?'}}, {'id': 1, 'name': 'image
generation', 'args': {'text': '<node-0>.text'}}]
New nodes: [{'id': 0, 'name': 'text generation', 'args': {'text': 'a comprehensive, easy-to-understand description of What Really Happens
When You Flush on an Airplane?'}}, {'id': 1, 'name': 'image generation', 'args': {'text': 'an illustrative image based on <node-0>.text'}}]

# REQUIREMENTS #:
1) Besides the argument values of 'text generation' and 'image generation', everything else (including the nodes' ids and names) must stay
the same;
2) The argument value can include reference to last node i's text output as <node-i>.text.
3) You must NOT add or remove any nodes.

Request: "I need to give a quick presentation for kindergarteners on 'Why is the sky blue?'. I don't really have time to sift through lots of
complex information and I need simple, straightforward explanations with a relevant image that kids can understand. Can you assist me
with that?"
Nodes: [{'id': 0, 'name': 'wikipedia simple search', 'args': {'text': 'Why is the sky blue'}}, {'id': 1, 'name': 'text summarization', 'args': {'text':
'<node-0>.text'}}, {'id': 2, 'nam
e': 'image generation', 'args': {'text': '<node-1>.text'}}]
New nodes:
```

Fig. 4: Argument value rewrite prompt. We present the full prompt used for rewriting the argument values of `text generation` and `image generation`.

by searching for all possible paths from t_o's input to its output. As for semantic alternative tools, we prompt GPT-4 to generate these for each tool in the toolset. For example, for the plan `image classification` $\rightarrow$ `text generation`, we first obtain alternative tools to each of them. For `image classification`, its syntactic alternative tools include `image captioning` and `visual question answering` as these tools' inputs both include one image and their outputs are a text – the same as `image classification`'s. In addition, GPT-4 identifies `object detection` as a semantic alternative to `image classification`. On the other hand, there are no human-verified alternative tools to `text generation`. Therefore, there are a total of 3 alternative plans to `image classification` $\rightarrow$ `text generation`.

C Planning agent

We present the full prompts used for multi-step JSON-format planning (Figure 6), step-by-step JSON-format planning (Figure 7, excluding details in the TOOL LIST which are the same as the ones in Figure 6) as well as code generation (Figure 8).

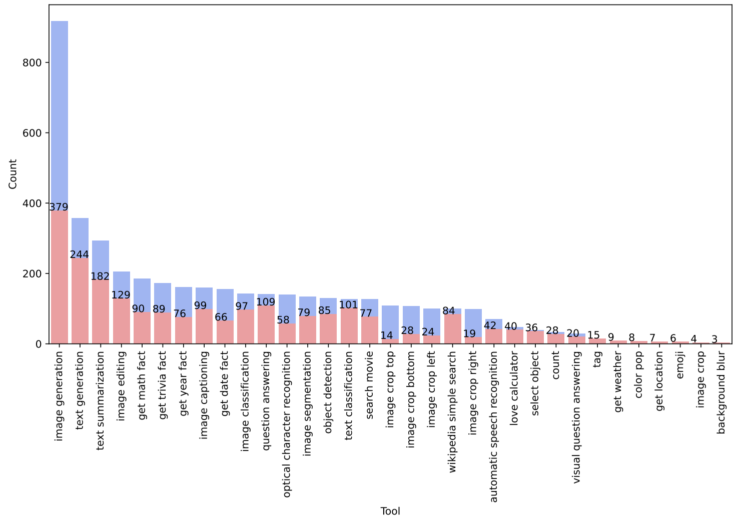

Fig. 5: Tool distribution before and after filtering.

Table 2: We present the tool-F1, argname-F1 and pass rate of models with various feedback, where P, V, and E represent parsing, verification, and execution feedback respectively. We use no feedback only (N/A) under multi-step planning and JSON-format language generation as the basis, while showing the Δ of those with other feedback combinations compared to no feedback.

	tool-F1					argname-F1					pass rate				
model	N/A	P	PV	PE	PVE	N/A	P	PV	PE	PVE	N/A	P	PV	PE	PVE
Llama-2-7b	27.37	2.41	-0.53	-0.18	-0.18	30.71	3.31	5.34	4.56	4.47	24.83	3.40	21.54	13.72	17.12
Llama-2-13b	40.30	1.97	-1.48	-0.80	-2.60	43.30	1.77	5.72	4.86	5.06	37.30	0.79	30.73	33.79	24.72
Mixtral-8x7B	65.06	1.73	0.88	0.15	2.75	73.00	-0.49	1.12	-0.14	0.85	69.61	6.12	16.44	15.08	16.89
Gemini-pro	68.57	0.80	1.98	0.69	0.76	72.79	0.58	2.58	2.47	3.30	73.92	3.40	16.67	17.46	20.07
GPT-3.5-turbo-0125	79.83	0.68	0.03	-2.11	-1.88	83.94	0.92	1.57	0.00	0.06	88.44	1.02	7.71	8.28	7.94
GPT-4-0125-preview	88.96	-0.50	-1.10	-0.26	-1.42	89.88	-0.07	-0.25	0.41	0.25	97.39	0.34	1.47	-0.91	2.49

D Additional plan evaluation results

Apart from the three main metrics in the main paper, we have also evaluated all six large language models on 10+ other metrics. We report these additional evaluation results below.

D.1 No feedback

In the main paper, we present the results of models with verification and/or execution of feedback (on top of parsing feedback) using the experiment with parsing (P) feedback as a baseline. Here, we report the results using the experiment with no feedback at all as the baseline in Table 2. We see that our main takeaway remains the same with this change: feedback helps improve models' argname-F1 by a small amount and pass rate by a lot, although it can lead to

Table 3: argvalue-F1. We present the argvalue-F1 of step-by-step and multi-step planning with JSON-format generation and different types of feedback.

		argvalue-F1			
model	strategy	P	PV	PE	PVE
Llama-2-7b	step-by-step	4.63	8.28	9.68	9.57
	multi-step	10.34	9.88	9.47	10.57
Llama-2-13b	step-by-step	7.10	11.30	12.59	12.64
	multi-step	15.39	17.11	15.84	16.71
Mixtral-8x7B	step-by-step	20.44	24.32	21.77	21.69
	multi-step	36.45	36.70	35.70	36.73
Gemini-pro	step-by-step	32.28	27.81	32.22	31.37
	multi-step	37.22	39.89	36.30	38.33
GPT-3.5-turbo-0125	step-by-step	29.58	28.32	23.61	23.24
	multi-step	45.64	46.54	45.15	45.56
GPT-4-0125-preview	step-by-step	47.37	46.91	34.49	34.84
	multi-step	51.02	51.08	51.70	51.99

Table 4: edge-F1. We present the edge-F1 of step-by-step and multi-step planning with JSON-format generation and different types of feedback.

		edge-F1			
model	strategy	P	PV	PE	PVE
Llama-2-7b	step-by-step	1.61	2.35	3.98	3.37
	multi-step	12.44	11.61	12.10	11.27
Llama-2-13b	step-by-step	5.74	6.22	6.96	8.22
	multi-step	23.27	23.98	24.00	23.58
Mixtral-8x7B	step-by-step	15.41	21.88	24.00	24.77
	multi-step	55.72	53.10	53.08	53.52
Gemini-pro	step-by-step	41.39	17.86	45.82	45.08
	multi-step	54.98	56.63	53.60	55.22
GPT-3.5-turbo-0125	step-by-step	31.37	27.23	39.40	39.72
	multi-step	69.52	71.03	67.98	69.05
GPT-4-0125-preview	step-by-step	73.68	72.67	68.28	68.12
	multi-step	78.80	78.79	79.47	79.60

a small decrease in tool-F1. We additionally observe the improvement of verification and/or execution feedback on pass rate is larger than that of parsing feedback.

D.2 Step-level metrics

Besides tool-F1 and argname-F1, we also report the following step-level metrics: argvalue-F1 (Table 3), edge-F1 (Table 4), and normalized edit distance (Table 5). We adapted TaskBench's [1] implementation of these metrics on our benchmark. We caution readers about argvalue-F1 as it is computed based on exact matching to one groundtruth value even though there can be multiple valid values.

D.3 Plan-level accuracy

Since step-level metrics do not take into account the ordering of the predicted tools, we additionally include plan-level accuracy to evaluate the whole plan's

Table 5: Normalized edit distance. We present the normalized edit distance of step-by-step and multi-step planning with JSON-format generation and different types of feedback.

		Normalized edit distance ↓			
model	strategy	P	PV	PE	PVE
Llama-2-7b	step-by-step	80.39	75.24	76.00	74.55
	multi-step	61.14	64.43	62.82	63.12
Llama-2-13b	step-by-step	72.81	68.57	68.60	67.84
	multi-step	47.57	48.69	49.63	49.73
Mixtral-8x7B	step-by-step	60.81	56.28	56.86	56.78
	multi-step	23.97	25.97	26.64	26.26
Gemini-pro	step-by-step	36.23	47.89	34.70	36.00
	multi-step	28.18	27.34	25.96	24.77
GPT-3.5-turbo-0125	step-by-step	51.46	52.38	47.93	47.44
	multi-step	16.08	15.55	17.44	17.86
GPT-4-0125-preview	step-by-step	14.26	14.70	16.92	16.62
	multi-step	10.96	11.39	10.59	10.81

Table 6: Plan accuracy

	Plan accuracy	(tool)				(tool+argname)			
model	strategy	P	PV	PE	PVE	P	PV	PE	PVE
Llama-2-7b	step-by-step	1.13	2.27	3.29	3.29	1.13	2.27	3.29	3.29
	multi-step	4.20	3.40	2.95	4.20	2.95	3.29	2.04	3.51
Llama-2-13b	step-by-step	1.25	3.17	3.74	4.99	1.13	3.17	3.74	4.99
	multi-step	11.90	13.83	10.88	12.13	9.52	13.27	9.98	11.79
Mixtral-8x7B	step-by-step	9.41	14.63	14.06	14.97	9.41	14.63	14.06	14.97
	multi-step	45.80	45.12	45.12	45.35	45.12	45.01	44.90	45.24
Gemini-pro	step-by-step	24.83	10.66	30.27	28.57	24.38	10.66	30.16	28.57
	multi-step	41.84	42.18	40.70	42.40	40.48	42.18	40.59	42.40
GPT-3.5-turbo-0125	step-by-step	19.27	14.97	18.59	19.16	19.27	14.97	18.59	19.16
	multi-step	59.64	60.20	57.48	58.39	59.52	60.20	57.48	58.39
GPT-4-0125-preview	step-by-step	61.68	60.88	51.93	53.17	61.68	60.88	51.93	53.17
	multi-step	70.63	69.50	71.43	70.63	70.63	69.50	71.43	70.63

correctness (Table 6). We highlight two main variants of plan accuracy in Table 6, where the first one considers a list of tool names as a plan and the second considers a list of (tool name, argument names) tuples as a plan. As there could be multiple valid plans of the same query, we have also included the Δ in plan accuracy considering alternative plans in Table 7 and shown that our set of alternative plans can recover 1-5% examples where the models could have output potential valid plans different from the one human-verified groundtruth plan. Finally, we also present the strictest form of plan accuracy, which considers a list of tool names, argument names and values as a plan in Table 8. We note that exact matching gives us (Table 8 left) extremely low scores while using entailment in the case of text values – if the predicted argument text entails the label text – gives us more reasonable scores (Table 8 right).

Additionally, we also include the plan accuracy of models across different numbers of tools with multi-step and step-by-step planning respectively in Tables 9 and 10. Under multi-step planning, we find that most models experience a drop

Table 7: Δ in plan accuracy considering alternative plans.

Δ in plan accuracy		(tool)				(tool+argname)			
model	strategy	P	PV	PE	PVE	P	PV	PE	PVE
Llama-2-7b	step-by-step	0.00	0.11	0.11	0.11	0.00	0.11	0.11	0.11
	multi-step	0.79	0.34	0.68	0.57	0.00	0.11	0.11	0.23
Llama-2-13b	step-by-step	0.57	0.57	0.68	0.91	0.45	0.57	0.68	0.91
	multi-step	1.36	1.47	1.47	1.47	0.91	1.36	1.25	1.25
Mixtral-8x7B	step-by-step	0.79	2.15	1.93	2.04	0.79	1.93	1.93	1.93
	multi-step	4.08	3.40	3.74	2.83	3.40	3.40	3.29	2.61
Gemini-pro	step-by-step	1.36	2.83	2.49	1.93	1.36	2.83	2.38	1.93
	multi-step	3.74	2.83	4.65	3.51	3.40	2.83	4.65	3.51
GPT-3.5-turbo-0125	step-by-step	1.02	0.34	1.02	0.68	1.02	0.34	1.02	0.68
	multi-step	3.17	3.06	3.40	3.74	3.17	3.06	3.40	3.74
GPT-4-0125-preview	step-by-step	2.15	1.81	2.95	3.06	2.15	1.81	2.95	3.06
	multi-step	1.81	1.81	1.59	1.59	1.81	1.81	1.59	1.59

Table 8: Plan accuracy considering argument values

Plan accuracy (tool+argname+argvalue)		exact matching				entailment			
model	strategy	P	PV	PE	PVE	P	PV	PE	PVE
Llama-2-7b	step-by-step	0.57	1.02	1.81	1.59	0.91	1.81	2.95	2.38
	multi-step	0.57	0.34	0.23	0.57	1.02	1.59	0.68	1.59
Llama-2-13b	step-by-step	0.57	1.70	2.04	2.27	0.91	2.49	2.83	3.51
	multi-step	2.04	2.72	2.38	2.49	5.44	7.48	5.78	6.24
Mixtral-8x7B	step-by-step	2.72	5.44	3.51	3.51	6.12	9.86	7.03	7.37
	multi-step	9.75	10.09	9.52	10.77	28.00	29.14	28.68	29.48
Gemini-pro	step-by-step	7.03	5.78	7.48	6.58	15.42	9.52	17.12	15.19
	multi-step	8.39	11.34	9.07	11.45	24.15	27.89	24.83	27.66
GPT-3.5-turbo-0125	step-by-step	6.46	5.33	2.38	2.72	12.93	10.20	7.14	8.05
	multi-step	13.61	14.29	13.61	14.06	34.81	36.85	34.92	35.83
GPT-4-0125-preview	step-by-step	11.68	11.00	6.35	6.24	34.35	32.65	19.73	20.29
	multi-step	14.85	14.97	15.19	15.53	41.04	40.70	43.20	42.97

in plan accuracy as the number of tools in the plans increases. Interestingly, the smaller models like LLama-7b and 13b exhibit a slightly different trend, achieving a higher number on 2-tool examples than on 1-tool ones (Table 9. One plausible explanation is that these models might not fully understand the user request and tend to output 2-tool plans more often. Surprisingly, GPT-4 also scores higher on 2-tools examples than 1-tool ones but with a much smaller gap. On the other hand, under step-by-step planning, we see that all models suffer from an even more drastic drop in plan accuracy as the number of tools required increases (Table 10). **This suggests that step-by-step planning might not scale well to more complex tasks that require a large number of tools/actions.**

D.4 Code-specific metrics: AST accuracy and CodeBLEU

To evaluate code generation properly, we have also included code-specific metrics such as AST accuracy and CodeBLEU (Table 11). AST accuracy measures if the

Table 9: Plan accuracy by number of tools with *multi-step* planning

Plan accuracy		(tool)				(tool + argname)			
model	# of tools	P	PV	PE	PVE	P	PV	PE	PVE
Llama-2-7b	1	1.43	4.29	1.43	1.43	1.43	4.29	1.43	1.43
	2	10.06	9.43	6.92	9.43	8.18	9.43	5.66	7.55
	3	3.06	1.84	2.14	3.22	1.84	1.68	1.23	2.76
Llama-2-13b	1	7.14	18.57	10.00	8.57	7.14	17.14	10.00	8.57
	2	18.87	23.90	16.98	18.24	16.98	23.27	15.09	17.61
	3	10.72	10.87	9.49	11.03	7.96	10.41	8.73	10.72
Mixtral-8x7B	1	70.00	71.43	71.43	71.43	68.57	71.43	71.43	71.43
	2	55.97	55.97	55.97	57.86	55.97	55.35	55.97	57.23
	3	40.74	39.66	39.66	39.51	39.97	39.66	39.36	39.51
Gemini-pro	1	65.71	74.29	74.29	78.57	65.71	74.29	74.29	78.57
	2	49.06	50.31	49.69	52.83	48.43	50.31	49.06	52.83
	3	37.52	36.75	34.92	35.99	35.83	36.75	34.92	35.99
GPT-3.5-turbo-0125	1	74.29	75.71	71.43	71.43	74.29	75.71	71.43	71.43
	2	72.96	72.33	69.81	70.44	72.96	72.33	69.81	70.44
	3	54.82	55.59	52.99	54.06	54.67	55.59	52.99	54.06
GPT-4-0125-preview	1	78.57	78.57	81.43	81.43	78.57	78.57	81.43	81.43
	2	80.50	79.25	78.62	78.62	80.50	79.25	78.62	78.62
	3	67.38	66.16	68.61	67.53	67.38	66.16	68.61	67.53

AST tree of the predicted code is the same as the label code, whereas CodeBLEU measures the similarity of the predicted code to the reference code. We find that feedback, especially verification feedback, can help improve models' AST accuracy but not necessarily CodeBLEU scores.

D.5 Efficiency

Besides models' planning performance, we also kept track of their token usage (Table 13) and numbers of conversation turns (Table 12). As expected, step-by-step planning generally requires more conversation turns and more tokens than multi-step planning. Similarly, feedback also increases token usage.

E Evaluation of plan execution outputs

E.1 Human evaluation

Since *m&m*'s consists of open-ended queries, which do not always have one single final answer, it is challenging to evaluate the execution results of the plans automatically. Thus, we resort to human evaluation of a small subset of 85 examples with reasonable execution results. Our manual evaluation reveals that GPT-4 achieves the best execution accuracy with multi-step planning and JSON-format generation compared to step-by-step planning or code generation (Table 14).

Table 10: Plan accuracy by number of tools with *step-by-step* planning

Plan accuracy		(tool)				(tool + argname)			
model	# of tools	P	PV	PE	PVE	P	PV	PE	PVE
Llama-2-7b	1	14.29	24.29	34.29	32.86	14.29	24.29	34.29	32.86
	2	0.00	0.63	2.52	3.14	0.00	0.63	2.52	3.14
	3	0.00	0.31	0.15	0.15	0.00	0.31	0.15	0.15
Llama-2-13b	1	11.43	32.86	31.43	35.71	10.00	32.86	31.43	35.71
	2	1.89	0.63	5.03	6.29	1.89	0.63	5.03	6.29
	3	0.00	0.61	0.46	1.38	0.00	0.61	0.46	1.38
Mixtral-8x7B	1	40.00	67.14	51.43	52.86	40.00	67.14	51.43	52.86
	2	22.64	30.19	28.93	32.70	22.64	30.19	28.93	32.70
	3	2.91	5.21	6.43	6.58	2.91	5.21	6.43	6.58
Gemini-pro	1	52.86	78.57	62.86	55.71	52.86	78.57	62.86	55.71
	2	36.48	10.69	42.77	42.14	35.85	10.69	42.77	42.14
	3	18.99	5.97	23.74	22.36	18.53	5.97	23.58	22.36
GPT-3.5-turbo-0125	1	67.14	72.86	32.86	31.43	67.14	72.86	32.86	31.43
	2	28.93	18.87	37.74	37.74	28.93	18.87	37.74	37.74
	3	11.79	7.81	12.40	13.32	11.79	7.81	12.40	13.32
GPT-4-0125-preview	1	84.29	82.86	80.00	84.29	84.29	82.86	80.00	84.29
	2	70.44	70.44	71.70	72.96	70.44	70.44	71.70	72.96
	3	57.12	56.20	44.10	45.02	57.12	56.20	44.10	45.02

Table 11: Code-specific metrics. We present the AST accuracy and CodeBLEU score of models under multi-step planning with code generation with or without feedback.

	AST accuracy				CodeBLEU			
model	P	PV	PE	PVE	P	PV	PE	PVE
Llama-2-7b	0.00	0.00	0.00	0.00	22.64	21.28	17.58	21.19
Llama-2-13b	0.11	0.23	0.00	0.00	29.96	27.09	20.29	27.62
Mixtral-8x7B	2.04	3.06	4.22	2.30	54.17	48.48	53.01	47.21
Gemini-pro	3.85	5.33	3.74	4.54	62.37	61.13	59.00	59.18
GPT-3.5-turbo-0125	3.29	4.76	3.29	4.42	60.79	60.32	58.96	59.99
GPT-4-0125-preview	4.31	5.10	4.42	5.33	68.52	68.37	68.68	68.51

E.2 Automatic evaluation

Nevertheless, as human evaluation is not scalable, we have also implemented automatic evaluation, which uses the groundtruth plans' final outputs as the golden answers and compares the predicted plans' results against them. Our implementation invokes different metrics (all in [0,1]) based on the outputs' modality: cosine similarity with SentenceBERT embeddings for texts, and CLIP embeddings for images and Average Precision for predicted objects. We report the average accuracy on 210 queries with plans that yield good and deterministic outputs in Tables 15 and 16.

Similar to our planning evaluation, most models achieve the best execution accuracy in multi-step planning with JSON format generation except for LLama-7b and Gemini-pro, where step-by-step planning leads to higher execution accuracy (Table 15). In addition, we also observe that verification and/or execution feedback do lead to some improvement (up to 10+%) in execution accuracy

Table 12: Average turn count. We present the average number of conversation turns in step-by-step and multi-step planning with JSON-format generation and different types of feedback.

		Average # of turns				
model	strategy	N/A	P	PV	PE	PVE
Llama-2-7b	step-by-step	2.00	3.54	4.03	3.26	3.52
	multi-step	1.00	1.10	2.18	1.95	1.99
Llama-2-13b	step-by-step	2.87	2.87	3.09	3.06	2.99
	multi-step	1.00	1.04	1.98	1.91	1.97
Mixtral-8x7B	step-by-step	2.98	6.37	5.55	6.02	6.09
	multi-step	1.00	1.14	2.43	2.74	2.81
Gemini-pro	step-by-step	2.31	3.01	2.28	3.67	3.78
	multi-step	1.00	1.20	1.84	1.80	1.88
GPT-3.5-turbo-0125	step-by-step	2.40	3.39	4.10	5.43	5.30
	multi-step	1.00	1.02	1.36	1.46	1.62
GPT-4-0125-preview	step-by-step	3.22	3.52	3.51	3.59	3.59
	multi-step	1.00	1.00	1.05	1.06	1.07

Table 13: Average number of input and output tokens

		Avg # of input tokens					Avg # of output tokens				
model	strategy	N/A	P	PV	PE	PVE	N/A	P	PV	PE	PVE
Llama-2-7b	step-by-step	5497.25	20627.60	22021.08	14356.79	13562.25	108.54	659.02	673.01	436.63	432.34
	multi-step	2184.19	3065.88	10215.74	6792.83	8570.81	273.65	320.95	735.02	478.79	636.73
Llama-2-13b	step-by-step	13084.77	14793.73	13962.84	11498.10	13025.18	535.74	620.00	495.34	446.56	489.17
	multi-step	2184.19	2651.22	8141.48	7375.54	8309.38	326.91	345.01	738.19	648.41	753.93
Gemini-pro	step-by-step	5661.28	7651.78	5653.98	10136.36	10560.46	115.70	171.22	96.98	216.03	232.53
	multi-step	2184.19	3062.00	4962.19	4786.80	5022.53	86.12	155.05	219.64	216.77	225.45
GPT-3.5-turbo-0125	step-by-step	5891.36	8938.04	11693.37	16497.09	15966.33	109.61	189.53	207.51	317.43	318.30
	multi-step	2184.19	2247.54	3199.10	3502.05	4017.90	96.24	99.47	136.24	149.94	166.76
GPT-4-0125-preview	step-by-step	8046.55	8852.87	8832.17	9601.61	9618.19	166.17	172.37	171.03	235.51	236.76
	multi-step	2184.19	2184.19	2318.98	2331.06	2354.78	102.28	103.49	110.55	107.74	111.09

compared to parsing feedback only, with only one exception in GPT-4 with execution feedback where the execution accuracy is comparable but not better (Table 16). Overall, these results suggest our execution output evaluation aligns well with our planning evaluation, providing further evidence to support our findings on the positive effects of multi-step planning, JSON-format generation and feedback.

References

1. Shen, Y., Song, K., Tan, X., Zhang, W., Ren, K., Yuan, S., Lu, W., Li, D., Zhuang, Y.: Taskbench: Benchmarking large language models for task automation. arXiv preprint arXiv:2311.18760 (2023)

Table 14: Human evaluation of execution outputs. We present the execution accuracy of GPT-4 and Mixtral-8x7B on a selected subset of 85 examples across different setups, including step-by-step and multi-step planning, with JSON-format and code generation, and different types of feedback.

model	strategy	format	feedback	accuracy
Mixtral-8x7B	multi-step	JSON	P	42.94 ± 1.76
GPT-4-0125-preview	step-by-step	JSON	P	49.41 ± 1.18
GPT-4-0125-preview	multi-step	Code	P	61.18 ± 0.0
GPT-4-0125-preview	multi-step	JSON	PVE	64.12 ± 2.94
GPT-4-0125-preview	multi-step	JSON	P	70.00 ± 6.47

Table 15: Automatic evaluation of execution outputs. We present the execution accuracy of models in step-by-step and multi-step planning and with JSON-format and code generation.

model	strategy	format	accuracy
Llama-2-7b	step-by-step	JSON	10.52
	multi-step	JSON	7.54
	multi-step	code	1.45
Llama-2-13b	step-by-step	JSON	9.17
	multi-step	JSON	12.27
	multi-step	code	7.43
Mixtral-8x7B	step-by-step	JSON	34.16
	multi-step	JSON	42.28
	multi-step	code	34.03
Gemini-pro	step-by-step	JSON	45.74
	multi-step	JSON	44.25
	multi-step	code	34.28
GPT-3.5-turbo-0125	step-by-step	JSON	38.36
	multi-step	JSON	49.47
	multi-step	code	42.85
GPT-4-0125-preview	step-by-step	JSON	50.86
	multi-step	JSON	60.51
	multi-step	code	54.49

Table 16: Automatic evaluation of execution outputs with feedback. We present the execution accuracy of models in multi-step planning with various feedback.

model	P	PV	PE	PVE
Llama-2-7b	7.54	13.54	8.90	8.51
Llama-2-13b	12.27	25.44	22.58	16.29
Mixtral-8x7B	42.28	45.57	46.19	42.92
Gemini-pro	44.25	51.72	49.47	55.18
GPT-3.5-turbo-0125	49.47	55.23	55.12	55.26
GPT-4-0125-preview	60.51	60.72	59.07	61.73

TOOL LIST #:
text generation: It takes an input text prompt and outputs a text that is most likely to follow the input text. Its input includes text, and output includes text.
text summarization: it takes a paragraph of text and summarizes into a few sentences. Its input includes text, and output includes text.
text classification: It takes a text and classifies it into a category in the model's vocabulary (e.g. positive or negative based on its sentiment). Its input includes text, and output includes text.
question answering: It takes a text and a question, and outputs an answer to that question based on the text. Its input includes text, question, and output includes text.
image generation: It takes a text prompt and generates an image that matches the text description. Its input includes text, and output includes image.
image captioning: It takes an image and generates a text caption of the image. Its input includes image, and output includes text.
optical character recognition: It takes an image and outputs recognized texts in the image. Its input includes image, and output includes text.
image classification: It takes an image and classifies the subject in the image into a category such as cat or dog. Its input includes image, and output includes text.
image editing: It takes an image and a text prompt and outputs a new image based on the text. Its input includes image, prompt, and output includes image.
object detection: It takes an image and outputs rectangular bounding boxes of objects detected in the image. Its input includes image, and output includes image, objects.
image segmentation: It takes an image, segments it into different parts, and outputs segmentation masks of any shape for the parts. Its input includes image, and output includes image, objects.
automatic speech recognition: It takes an audio file and produces a transcription of the audio. Its input includes audio, and output includes text.
visual question answering: It takes an image and a question about the image, and generates an answer to the question. Its input includes image, question, and output includes text.
image crop: It takes an image and 4 numbers representing the coordinates of a bounding box and crops the image to the region within the box. Its input includes image, object, and output includes image.
image crop left: It takes an image, crops and keeps the left part of the image. Its input includes image, and output includes image.
image crop right: It takes an image, crops and keeps the right part of the image. Its input includes image, and output includes image.
image crop top: It takes an image, crops and keeps the top part of the image. Its input includes image, and output includes image.
image crop bottom: It takes an image, crops and keeps the bottom part of the image. Its input includes image, and output includes image.
background blur: It takes an image and one or multiple objects in the foreground, and returns an image where the backgroud is blurred. Its input includes image, object, and output includes image.
color pop: It takes an image and one or multiple objects, and returns an image where only the object is colored and the rest is black and white. Its input includes image, object and output includes image.
count: It takes a list of objects and returns the count of the objects. Its input includes objects, and output includes number.
tag: It takes an image and a list of objects with their bounding boxes and classes, and tags all the objects Its input includes image, objects, and output includes image.
select object: It takes a list of objects, and selects the object based on the input object name. Its input includes objects, object_name, and output includes object.
emoji: It takes an image and the bounding box coordinates of one or multiple objects, and replaces the object with an emoji (e.g. angry/flushed/crying/dizzy/sleepy/grimacing/kissing/smiling_face, alien, ghost, goblin etc). Its input includes image, object, emoji, and output includes image.
get date fact: It provides interesting facts about dates. Its input includes date, and output includes text.
get year fact: It provides interesting facts about years. Its input includes year, and output includes text.
get math fact: It provides interesting math facts about numbers. Its input includes number, and output includes text.
get trivia fact: It provides interesting trivia facts about number. Its input includes number, and output includes text.
love calculator: Enter your name and the name of your partner/lover/crush to find Love compatibility & chances of successful love relationship. Its input includes first_name, second_name, and output includes number.
get location: Convert a city name or address to geographical coordinates using OpenStreetMap's Nominatim API. Its input includes city, and output includes lon, lat.
search movie: Retrieve basic movie information, including title, year, genre, and director. Its input includes movie_title, movie_year, and output includes text.
get weather: Provides weather forecast data based on specific geographical coordinates. Its input includes lon, lat, and output includes objects.
wikipedia simple search: Perform a basic search query on Wikipedia to retrieve a summary of the most relevant page. Its input includes text, and output includes text.

GOAL #: Based on the above tools, I want you to generate the task nodes to solve the # USER REQUEST #. The format must be in a strict JSON format, like: {"nodes": [{"id": an integer id of the tool, starting from 0, "name": "tool name must be from # TOOL LIST #", "args": { a dictionary of arguments for the tool. Either original text, or user-mentioned filename, or tag '<node-j>.text' (start from 0) to refer to the text output of the j-th node. }}]}

REQUIREMENTS #:
1. the generated tool nodes can resolve the given user request # USER REQUEST # perfectly. Tool name must be selected from # TOOL LIST #;
2. The arguments of a tool must be the same number, modality, and format specified in # TOOL LIST #;
3. Use as few tools as possible.

EXAMPLE #:
USER REQUEST #: "Based on reading the article titled 'Would you rather have an Apple Watch - or a BABY?', generate an extended paragraph on the topic."
RESULT #: {"nodes": [{"id": 0, "name": "text generation", "args": {"text": "an extended paragraph on the topic: Would you rather have an Apple Watch - or a BABY?"}}]}
EXAMPLE #:
USER REQUEST #: "Could you take the image, specifically 'image 17320.jpg', and adjust it so the green ball in the picture becomes blue, then describe for me what the resulting
image looks like?"
RESULT #: {"nodes": [{"id": 0, "name": "image editing", "args": {"image": "17320.jpg", "prompt": "change the green ball to blue"}}, {"id": 1, "name": "image captioning", "args: {"image": "<node-0>.image"}}]}
EXAMPLE #:
USER REQUEST #: "Could you provide a brief summary of the key points discussed in the audio file '1995-1826-0002.flac' about John Taylor and his interest in cotton? And then, can you also help me create a vivid illustration based on the key points?"
RESULT #: {"nodes": [{"id": 0, "name": "automatic speech recognition", "args": {"audio": "1995-1826-0002.flac"}}, {"id": 1, "name": "text summarization", "args": {"text": "<node-0>.text"}}, {"id": 2, "name": "image generation", "args": {"text": "a vivid illustration based on <node-1>.text"}}]}

USER REQUEST #: "I need to give a quick presentation for kindergarteners on 'Why is the sky blue?'. I don't really have time to sift through lots of complex information and I need simple, straightforward explanations with a relevant image that kids can understand. Can you assist me with that?"
Now please generate your result in a strict JSON format:
RESULT #:

Fig. 6: Multi-step planning prompt. We present the full prompt used for multi-step planning.

```
# TOOL LIST #:
text generation: It takes an input text prompt and outputs a text that is most likely to follow the input text. Its input includes text, and output
includes text.
text summarization: it takes a paragraph of text and summarizes into a few sentences. Its input includes text, and output includes text.
text classification: It takes a text and classifies it into a category in the model's vocabulary (e.g. positive or negative based on its sentiment).
Its input includes text, and output includes text.
question answering: It takes a text and a question, and outputs an answer to that question based on the text. Its input includes text,
question, and output includes text.
image generation: It takes a text prompt and generates an image that matches the text description. Its input includes text, and output
includes image.
......

# GOAL #: Based on the above tools, I want you to reason about how to solve the # USER REQUEST # and generate the actions step by step.

# REQUIREMENTS #:
1. The thoughts can be any free form texts to help with action generation;
2. The action must follow this JSON format strictly: {"id": an integer id of the tool, starting from 0, which should be the same as the id of the
ACTION "name": "tool name must be from # TOOL LIST #", "args": { a dictionary of
arguments for the tool. Either original text, or user-mentioned filename, or tag '<node-j>.text' (start from 0) to refer to the text output of the
j-th node. }};
3. The arguments of a tool must match the number, modality, and format of the tool's arguments specified in # TOOL LIST #.
# EXAMPLE #:
# USER REQUEST #: "Based on reading the article titled 'Would you rather have an Apple Watch - or a BABY?', generate an extended
paragraph on the topic."
# RESULT #:
THOUGHT 0: First, I need to perform text generation.
ACTION 0: {"id": 0, "name": "text generation", "args": {"text": "Would you rather have an Apple Watch - or a BABY?"}}

# EXAMPLE #:
# USER REQUEST #: "Could you take the image, specifically 'image 17320.jpg', and adjust it so the green ball in the picture becomes blue,
then describe for me what the resulting
image looks like?"
# RESULT #:
THOUGHT 0: First, I need to perform image editing.
ACTION 0: {"id": 0, "name": "image editing", "args": {"image": "17320.jpg", "prompt": "change the green ball to blue"}}
OBSERVATION 0: {'image': 'an image with a blue ball in it'}
THOUGHT 1: Based on the user query and OBSERVATION 0, then, I need to perform image captioning.
ACTION 1: {"id": 1, "name": "image captioning", "args": {"image": "<node-0>.image"}}

# EXAMPLE #:
# USER REQUEST #: "Could you provide a brief summary of the key points discussed in the audio file '1995-1826-0002.flac' about John
Taylor and his interest in cotton? And then, c
an you also help me create a vivid illustration based on the key points?"
# RESULT #:
THOUGHT 0: First, I need to perform automatic speech recognition.
ACTION 0: {"id": 0, "name": "automatic speech recognition", "args": {"audio": "1995-1826-0002.flac"}}
OBSERVATION 0: {'text': 'John Taylor, who had supported her through college, was interested in cotton.'}
THOUGHT 1: Based on the user query and OBSERVATION 0, then, I need to perform text summarization.
ACTION 1: {"id": 1, "name": "text summarization", "args": {"text": "<node-0>.text"}}
OBSERVATION 1: {'text': 'John Taylor was interested in cotton.'}
THOUGHT 2: Based on the user query and OBSERVATION 1, then, I need to perform image generation.
ACTION 2: {"id": 2, "name": "image generation", "args": {"text": "a vivid illustration based on <node-1>.text"}}

# USER REQUEST #: I came across a term - 'Juneteenth' in a book. To better comprehend the context, can I have a summarized information
about 'Juneteenth' along with a visual depi
ction of it?
Now please generate only THOUGHT 0 and ACTION 0 in RESULT:
# RESULT #:
```

Fig. 7: Step-by-step planning prompt. We present the full prompt used for step-by-step planning.

````
# TOOL LIST #:
text_generation(text) → text: It takes an input text prompt and outputs a text that is most likely to follow the input text.
text_summarization(text) → text: it takes a paragraph of text and summarizes into a few sentences.
text_classification(text) → text: It takes a text and classifies it into a category in the model's vocaburary (e.g. positive or negative based on its sentiment).
question_answering(text, question) → text: It takes a text and a question, and outputs an answer to that question based on the text.
image_generation(text) → image: It takes a text prompt and generates an image that matches the text description.
image_captioning(image) → text: It takes an image and generates a text caption of the image.
optical_character_recognition(image) → text: It takes an image and outputs recognized texts in the image.
image_classification(image) → text: It takes an image and classifies the subject in the image into a category such as cat or dog.
image_editing(image, prompt) → image: It takes an image and a text prompt and outputs a new image based on the text.
object_detection(image) → image, objects: It takes an image and outputs rectangular bounding boxes of objects detected in the image.
image_segmentation(image) → image, objects: It takes an image, segments it into different parts, and outputs segmentation masks of any shape for the parts.
automatic_speech_recognition(audio) → text: It takes an audio file and produces a transcription of the audio.
visual_question_answering(image, question) → text: It takes an image and a question about the image, and generates an answer to the question.
image_crop(image, object) → image: It takes an image and 4 numbers representing the coordinates of a bounding box and crops the image to the region within the box.
image_crop_left(image) → image: It takes an image, crops and keeps the left part of the image.
image_crop_right(image) → image: It takes an image, crops and keeps the right part of the image.
image_crop_top(image) → image: It takes an image, crops and keeps the top part of the image.
image_crop_bottom(image) → image: It takes an image, crops and keeps the bottom part of the image.
background_blur(image, object) → image: It takes an image and one or multiple objects in the foreground, and returns an image where the backgroud is blurred.
color_pop(image, object) → image: It takes an image and one or multiple objects, and returns an image where only the object is colored and the rest is black and white.
count(objects) → number: It takes a list of objects and returns the count of the objects.
tag(image, objects) → image: It takes an image and a list of objects with their bounding boxes and classes, and tags all the objects
select_object(objects, object_name) → object: It takes a list of objects, and selects the object based on the input object name.
emoji(image, object, emoji) → image: It takes an image and the bounding box coordinates of one or multiple objects, and replaces the object with an emoji (e.g. angry/flushed/crying/dizzy/sleepy/grimacing/kissing/smiling_face, alien, ghost,
goblin etc).
get_date_fact(date) → text: It provides interesting facts about dates.
get_year_fact(year) → text: It provides interesting facts about years.
get_math_fact(number) → text: It provides interesting math facts about numbers.
get_trivia_fact(number) → text: It provides interesting trivia facts about number.
love_calculator(first_name, second_name) → number: Enter your name and the name of your partner/lover/crush to find Love compatibility & chances of successful love relationship.
get_location(city) → lon, lat: Convert a city name or address to geographical coordinates using OpenStreetMap's Nominatim API.
search_movie(movie_title, movie_year) → text: Retrieve basic movie information, including title, year, genre, and director.
get_weather(lon, lat) → objects: Provides weather forecast data based on specific geographical coordinates.
wikipedia_simple_search(text) → text: Perform a basic search query on Wikipedia to retrieve a summary of the most relevant page.

# GOAL #: Based on the above tools, I want you to generate a python program to solve the # USER REQUEST #.

# REQUIREMENTS #:
1. the generated program can resolve the given user request # USER REQUEST # perfectly. The functions must be selected from # TOOL LIST #;
2. The arguments of a function must be the same number, modality, and format specified in # TOOL LIST #;
3. Use as few tools as possible.

# EXAMPLE #:
# USER REQUEST #: "Based on reading the article titled 'Would you rather have an Apple Watch - or a BABY?', generate an extended paragraph on the topic."
# RESULT #: ```python
def solve():
    output0 = text_generation(text="an extended paragraph on the topic: Would you rather have an Apple Watch - or a BABY?")
    result = {0: output0}
    return result
```
# EXAMPLE #:
# USER REQUEST #: "Could you take the image, specifically 'image 17320.jpg', and adjust it so the green ball in the picture becomes blue, then describe for me what the resulting image looks like?"
# RESULT #: ```python
def solve():
    output0 = image_editing(image="17320.jpg", prompt="change the green ball to blue")
    output1 = image_captioning(image=output0['image'])
    result = {0: output0, 1: output1}
    return result
```
# EXAMPLE #:
# USER REQUEST #: "Could you provide a brief summary of the key points discussed in the audio file '1995-1826-0002.flac' about John Taylor and his interest in cotton? And then, can you also help me create a vivid illustration based on the ke
y points?"
# RESULT #: ```python
def solve():
    output0 = automatic_speech_recognition(audio="1995-1826-0002.flac")
    output1 = text_summarization(text=f"{output0['text']}")
    output2 = image_generation(text=f"a vivid illustration based on {output1['text']}")
    result = {0: output0, 1: output1, 2: output2}
    return result
```

# USER REQUEST #: "I need to give a quick presentation for kindergarteners on 'Why is the sky blue?'. I don't really have time to sift through lots of complex information and I need simple, straightforward explanations with a relevant image that kids can understand. Can you assist me with that?"
Now please generate your program enclosed in ```python ```:
````

Fig. 8: Code generation prompt. We present the full prompt used for code generation.
````