Language Models As Agents

Melak Senay, Chuqi Jiang, Anthony Zhou
Presentation date : 10/24/24

Introduction

What does Language Models as Agents mean?

When we talk about language models as agents, we're referring to models that can:

- Make decisions
- Interact with external environments (web browsers, APIs, …)
- Perform complex tasks autonomously

Agenda

- Toolformer: Language Models Can Teach Themselves to Use Tools
- ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs
- ART: Automatic multi-step reasoning and tool-use for large language models
- LLM+P: Empowering Large Language Models with Optimal Planning Proficiency

WashU Medicine

Toolformer: Language Models Can Teach Themselves to Use Tools

Author: Timo Schick et al.

https://arxiv.org/abs/2302.04761

WashU Medicine

An example of ToolFormer

- What is ToolFormer?
  - A model trained to decide which APIs to call
  - Best incorporates the results into future prediction

API Name	Example Input	Example Output
Question Answering	Where was the Knights of Columbus founded?	New Haven, Connecticut
Wikipedia Search	Fishing Reel Types	Spin fishing > Spin fishing is distinguished between fly fishing and bait cast fishing by the type of rod and reel used. There are two types of reels used when spin fishing, the open faced reel and the closed faced reel.
Calculator	27 + 4 * 2	35
Calendar	ε	Today is Monday, January 30, 2023.
Machine Translation	sûreté nucléaire	nuclear safety

Table 1: Examples of inputs and outputs for all APIs used.

The New England Journal of Medicine is a registered trademark of [QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society] the MMS.

Out of 1400 participants, 400 (or [Calculator(400 / 1400) → 0.29] 29%) passed the test.

The name derives from "la tortuga", the Spanish word for [MT("tortuga") → turtle] turtle.

The Brown Act is California's law [WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.] that requires legislative bodies, like city councils, to hold their meetings open to the public.

Figure 1: Exemplary predictions of Toolformer. The model autonomously decides to call different APIs (from top to bottom: a question answering system, a calculator, a machine translation system, and a Wikipedia search engine) to obtain information that is useful for completing a piece of text.

Motivation

Why ToolFormer?

- LMs are not designed to handle specialized tasks
  - Temporal inferences
  - Low resource languages
- APIs can be used to aid the LM in places where it lacks knowledge
  - Calculations
  - Search

Motivation

Why not just a bigger model?

- The most powerful models fail
  - Impossible to be up to date at all times
- Successful model can decide which APIs to call
  - APIs enhance responses

Main Contribution

- Flexible Tool Integration
  - Uses diverse set of APIs depending on the context
- Selection Done Self-supervised
  - After a few examples, it can annotate a dataset

Method Design

Transforming C -> C* takes three steps:

1) Sampling potential API Calls
2) Execute the API Calls
3) Filter API Calls

Method Design

$$\mathrm{e}(c) = \texttt{<API>}\, a_c\, (i_c)\ \texttt{</API>}$$
$$\mathrm{e}(c, r) = \texttt{<API>}\, a_c\, (i_c)\ \rightarrow r\, \texttt{</API>}$$

API name

The New England Journal of Medicine is a registered trademark of **[QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society]** the MMS.

Out of 1400 participants, 400 (or **[Calculator(400 / 1400) → 0.29]** 29%) passed the test.

The name derives from "la tortuga", the Spanish word for **[MT("tortuga") → turtle]** turtle.

The Brown Act is California's law **[WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.]** that requires legislative bodies, like city councils, to hold their meetings open to the public.

Method Design

Your task is to add calls to a Question Answering API to a piece of text. The questions should help you get information required to complete the text. You can call the API by writing "[QA(question)]" where "question" is the question you want to ask. Here are some examples of API calls:

Input: Joe Biden was born in Scranton, Pennsylvania.

Output: Joe Biden was born in **[QA("Where was Joe Biden born?")]** Scranton, **[QA("In which state is Scranton?")]** Pennsylvania.

Input: Coca-Cola, or Coke, is a carbonated soft drink manufactured by the Coca-Cola Company.

Output: Coca-Cola, or **[QA("What other name is Coca-Cola known by?")]** Coke, is a carbonated soft drink manufactured by **[QA("Who manufactures Coca-Cola?")]** the Coca-Cola Company.

Input: x

Output:

$$p_i = p_M(\texttt{<API>} \mid P(\mathbf{x}), x_{1:i-1})$$

Method Design

- The model samples a position after 400 and makes an API call to calculate the percentage

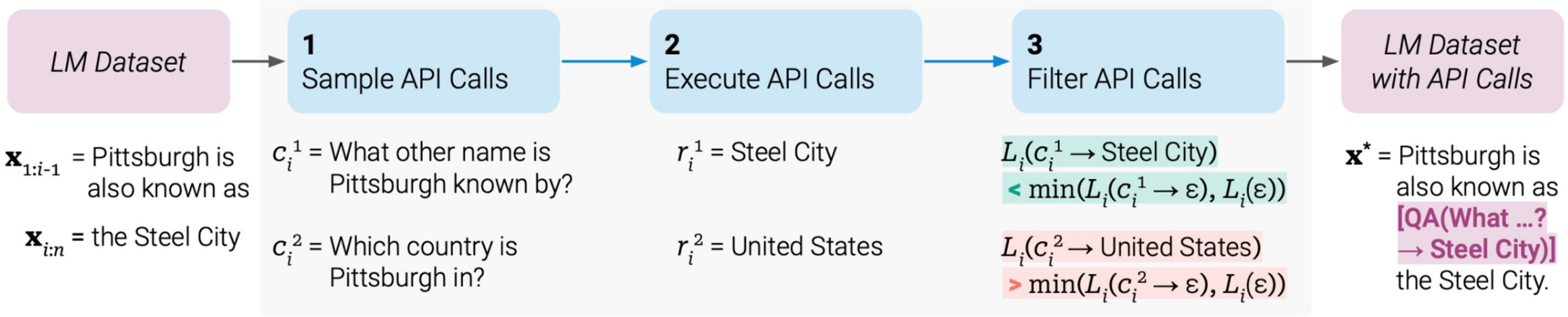
Out of 1400 participants, 400 (or [Calculator(400 / 1400) → 0.29] 29%) passed the test.

Method Design

- We can compare the loss of API Calls (more on this later)

$$\mathrm{e}(c) = \texttt{<API>}\, a_c\, (\, i_c\,)\ \texttt{</API>}$$

$$\mathrm{e}(c, r) = \texttt{<API>}\, a_c\, (\, i_c\,)\ \rightarrow r\, \texttt{</API>}$$



Method Design

Weighted Cross Entropy Loss:

$$L_i(\mathbf{z}) = -\sum_{j=i}^{n} w_{j-i} \cdot \log p_M(x_j \mid \mathbf{z}, x_{1:j-1})$$

$$L_i^+ = L_i(\mathrm{e}(c_i, r_i))$$

$$L_i^- = \min\left(L_i(\varepsilon), L_i(\mathrm{e}(c_i, \varepsilon))\right)$$

$$L_i^- - L_i^+ \geq \tau_f$$

Loss Example

Example	$L_i^- - L_i^+$	Useful
The Flodden Window (a war memorial dedicated to The Middleton Archers), in the Grade I-listed Church of St Leonard in Middleton is said to be the oldest war memorial in the United Kingdom. <API> **WikiSearch(War memorial Flodden) → Battle of Flodden > Commemoration > The stained-glass Flodden Window in Middleton Parish Church [...] was constructed by Sir Richard Assheton in memory of the Battle of Flodden and the archers from Middleton who fought in it.** </API> Sir Richard Assheton of Middleton (who built St Leonard) was granted knighthood [...]	5.49	✓
Note: The WL will be open on Friday, <API> **Calendar() → Today is Thursday, March 9, 2017.** </API> March 10, and Sunday, March 19 for regular hours.	2.11	✓
The Nile has an approximate length of <API> **QA(What is the approximate length of the Nile?) → 6,853 km** </API> 6,853 kilometers, the White Nile being its main source.	2.08	✓
If Venus had an atmosphere similar to Earth's then you would expect Venus' mean temperature to be 499 K (1.74 x 287) rather than 735 K which is <API> **Calculator(735 / 499) → 1.47** </API> 1.47 (735 / 499) times hotter than it should be.	1.59	✓
You are here: Home / Featured / Catch this fast train to success! <API> **WikiSearch(Fast train success) → Fast Train > It also peaked at #23 on the Canadian CHUM singles chart, on June 26, 1971. The success of this single established Myles Goodwyn as the band's main songwriter, and made it possible for April Wine to record a second album.** </API> Don't wait weeks and [...]	0.92	✗
Os Melhores Escolas em Jersey 2020 <API> **MT(Os Melhores Escolas em Jersey) → The Best Schools in Jersey** </API> On this page you can search for Universities, Colleges and Business schools in Jersey	0.70	✓
Enjoy these pictures from the <API> **Calendar() → Today is Friday, April 19, 2013.** </API> Easter Egg Hunt.	0.33	✓
85 patients (23%) were hospitalised alive and admitted to a hospital ward. Of them, <API> **Calculator(85 / 23) → 3.70** </API> 65% had a cardiac aetiology [...]	−0.02	✗
But hey, after the <API> **Calendar() → Today is Saturday, June 25, 2011.** </API> Disneyland fiasco with the fire drill, I think it's safe to say Chewey won't let anyone die in a fire.	−0.41	✗
The last time I was with <API> **QA(Who was last time I was with?) → The Last Time** </API> him I asked what he likes about me and he said he would tell me one day.	−1.23	✗

Table 10: Examples of API calls for different tools, sorted by the value of $L_i^- - L_i^+$ that is used as a filtering criterion. High values typically correspond to API calls that are intuitively useful for predicting future tokens.

Method Design

- C* has now been created
  - Now we must finetune the model on this new dataset
- Fine-tuning on C* allows the LM to decide when and how to use each tool (API) based on its own feedback.

Tools

- Question Answering
  - Factual Information lookup
- Calculator
  - Precise Arithmetic
- Wikipedia Search Engine
  - Get up to date information
- Machine Translation System
  - Language Translation
- Calendar
  - Returns current date

The New England Journal of Medicine is a registered trademark of **[QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society]** the MMS.

Out of 1400 participants, 400 (or **[Calculator(400 / 1400) → 0.29]** 29%) passed the test.

The name derives from "la tortuga", the Spanish word for **[MT("tortuga") → turtle]** turtle.

The Brown Act is California's law **[WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.]** that requires legislative bodies, like city councils, to hold their meetings open to the public.

Experiments

- Data: CCNet as the Language Modeling Dataset C
- Language Model M
  - GPT-J
- Fine-tuning
  - Finetune M on C.
  - Batch size of 128 with lr = 1E-5
- Comparison
  - Compare results with those from GPT 3 and OPT models
- Greedy Decoding

Experiments

Model	SQuAD	Google-RE	T-REx
GPT-J	17.8	4.9	31.9
GPT-J + CC	19.2	5.6	33.2
Toolformer (disabled)	22.1	6.3	34.9
Toolformer	**33.8**	**11.5**	**53.5**
OPT (66B)	21.6	2.9	30.1
GPT-3 (175B)	26.8	7.0	39.8

Table 3: Results on subsets of LAMA. Toolformer uses the question answering tool for most examples, clearly outperforming all baselines of the same size and achieving results competitive with GPT-3 (175B).

Model	ASDiv	SVAMP	MAWPS
GPT-J	7.5	5.2	9.9
GPT-J + CC	9.6	5.0	9.3
Toolformer (disabled)	14.8	6.3	15.0
Toolformer	**40.4**	**29.4**	**44.0**
OPT (66B)	6.0	4.9	7.9
GPT-3 (175B)	14.0	10.0	19.8

Table 4: Results for various benchmarks requiring mathematical reasoning. Toolformer makes use of the calculator tool for most examples, clearly outperforming even OPT (66B) and GPT-3 (175B).

Experiments

Model	Es	De	Hi	Vi	Zh	Ar
GPT-J	15.2	**16.5**	1.3	8.2	**18.2**	**8.2**
GPT-J + CC	15.7	14.9	0.5	8.3	13.7	4.6
Toolformer (disabled)	19.8	11.9	1.2	10.1	15.0	3.1
Toolformer	**20.6**	13.5	**1.4**	**10.6**	16.8	3.7
OPT (66B)	0.3	0.1	1.1	0.2	0.7	0.1
GPT-3 (175B)	3.4	1.1	0.1	1.7	17.7	0.1
GPT-J (All En)	24.3	27.0	23.9	23.3	23.1	23.6
GPT-3 (All En)	24.7	27.2	26.1	24.9	23.6	24.0

Table 6: Results on MLQA for Spanish (Es), German (De), Hindi (Hi), Vietnamese (Vi), Chinese (Zh) and Arabic (Ar). While using the machine translation tool to translate questions is helpful across all languages, further pretraining on CCNet deteriorates performance; consequently, Toolformer does not consistently outperform GPT-J. The final two rows correspond to models that are given contexts and questions in English.

Model	WebQS	NQ	TriviaQA
GPT-J	18.5	12.8	43.9
GPT-J + CC	18.4	12.2	45.6
Toolformer (disabled)	18.9	12.6	46.7
Toolformer	**26.3**	**17.7**	**48.8**
OPT (66B)	18.6	11.4	45.7
GPT-3 (175B)	29.0	22.6	65.9

Table 5: Results for various question answering dataset. Using the Wikipedia search tool for most examples, Toolformer clearly outperforms baselines of the same size, but falls short of GPT-3 (175B).

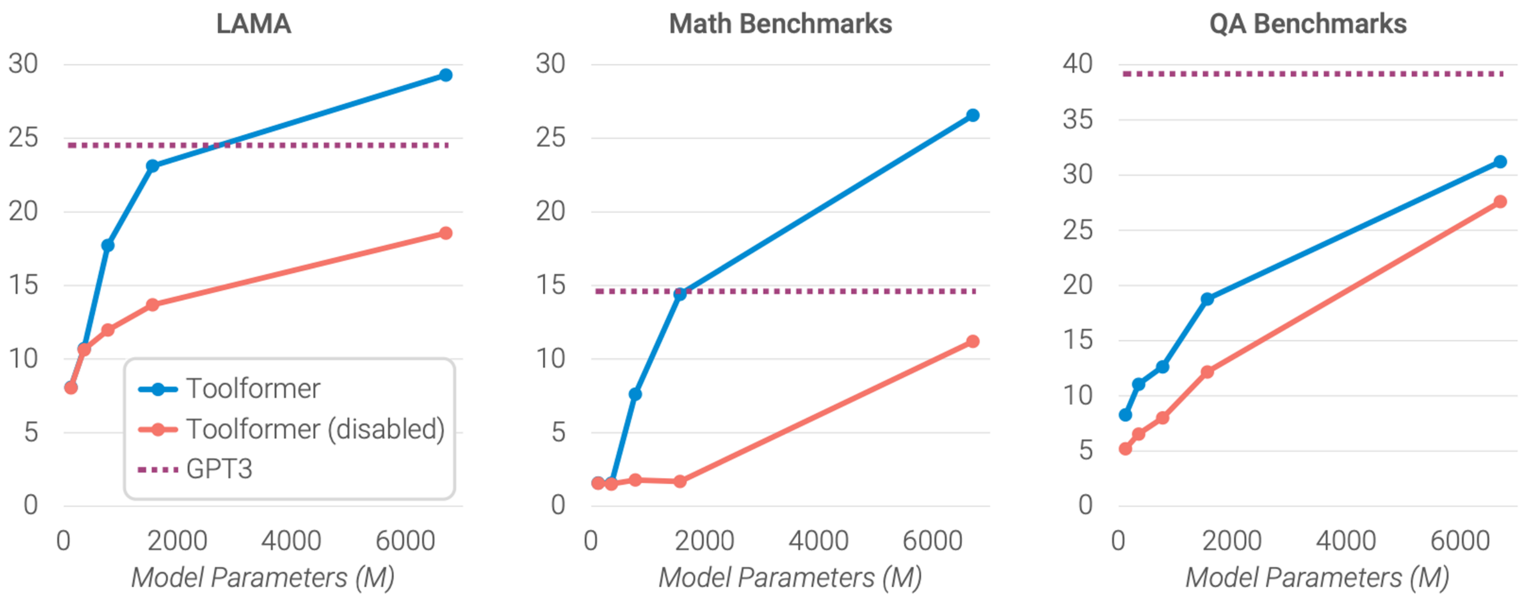
Model	TEMPLAMA	DATESET
GPT-J	13.7	3.9
GPT-J + CC	12.9	2.9
Toolformer (disabled)	12.7	5.9
Toolformer	**16.3**	**27.3**
OPT (66B)	14.5	1.3
GPT-3 (175B)	15.5	0.8

Table 7: Results for the temporal datasets. Toolformer outperforms all baselines, but does not make use of the calendar tool for TEMPLAMA.

Model	WikiText	CCNet
GPT-J	**9.9**	10.6
GPT-J + CC	10.3	**10.5**
Toolformer (disabled)	10.3	**10.5**

Table 8: Perplexities of different models on WikiText and our validation subset of CCNet. Adding API calls comes without a cost in terms of perplexity for language modeling without any API calls.

Analysis



LAMA
Math Benchmarks
QA Benchmarks
30
25
20
15
10
5
0
40
35
0
2000
4000
6000
Model Parameters (M)
Toolformer
Toolformer (disabled)
GPT3

Analysis

	T-REx				WebQS			
k	All	AC	NC	%	All	AC	NC	%
0	34.9	–	34.9	0.0	18.9	–	18.9	0.0
1	47.8	53.0	44.3	40.3	19.3	17.1	19.9	8.5
3	52.9	58.0	29.0	82.8	**26.3**	26.5	6.6	99.3
10	**53.5**	54.0	22.5	98.1	**26.3**	26.4	–	100.0

Table 9: Toolformer results on the T-REx subset of LAMA and on WebQS for different values of k used during decoding. Numbers shown are overall performance (All), performance on the subset where the model decides to make an API call (AC) and all remaining examples (NC), as well as the percentage of examples for which the model decides to call an API (%).

ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs

Author: Yujia Qin et al.

https://arxiv.org/abs/2307.16789

Motivation

- Instruction tuning focuses on basic language tasks
- This ignores the complex tool use domain (real time facts, etc.)

Motivation

- Current solutions have inherent limitations:

1) Limited APIs
    a) Fail to involve real world APIs like REST APIs
2) Constrained scenario
    a) Only works when instructions are confined to a single tool
3) Inferior Planning and reasoning.
    a) Ex: using CoT only. This doesn’t represent the complexity of modern LMs

Method Design

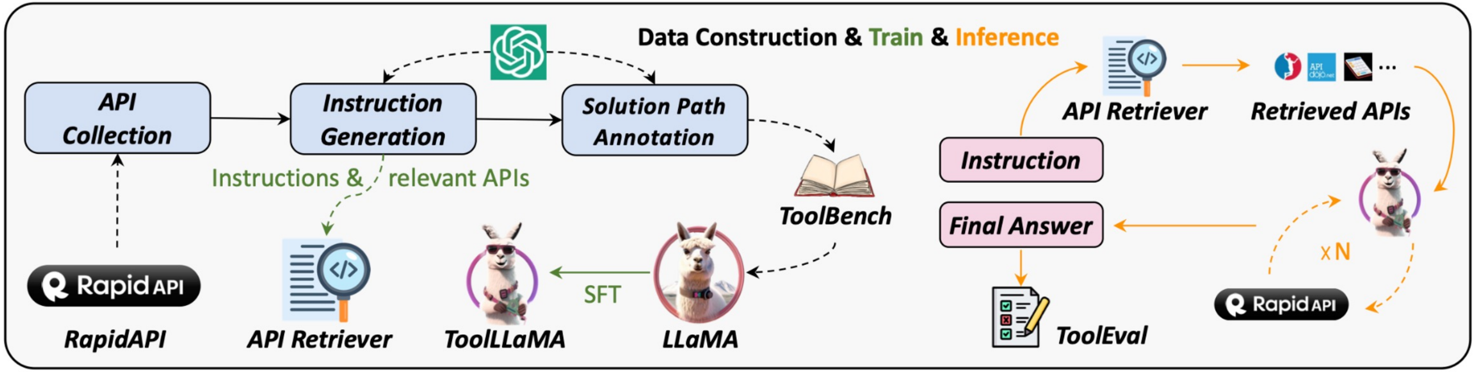


Figure 1: Three phases of constructing ToolBench and how we train our API retriever and ToolLLaMA. During inference of an instruction, the API retriever recommends relevant APIs to ToolLLaMA, which performs multiple rounds of API calls to derive the final answer. The whole reasoning process is evaluated by ToolEval.

Method Design

Resource	ToolBench (this work)	APIBench (Patil et al., 2023)	API-Bank (Li et al., 2023a)	ToolAlpaca (Tang et al., 2023)	ToolBench (Xu et al., 2023b)
Real-world API?	✓	✗	✓	✗	✓
Real API Call&Response?	✓	✗	✓	✗	✓
Multi-tool Scenario?	✓	✗	✗	✗	✗
API Retrieval?	✓	✓	✗	✗	✓
Multi-step Reasoning?	✓	✗	✓	✓	✓
Number of tools	**3451**	3	53	400	8
Number of APIs	**16464**	1645	53	400	232
Number of Instances	**126486**	17002	274	3938	2746
Number of Real API Calls	**469585**	0	568	0	3926
Avg. Reasoning Traces	4.0	1.0	2.1	1.0	**5.9**

Table 1: A comparison of our ToolBench to notable instruction tuning dataset for tool learning.

Dataset Construction: API Collection

instruction generation:

- API documentation is fed into the model.
- Sampled APIs used to create instructions.
- Relevant APIs are identified to complete complex tasks.

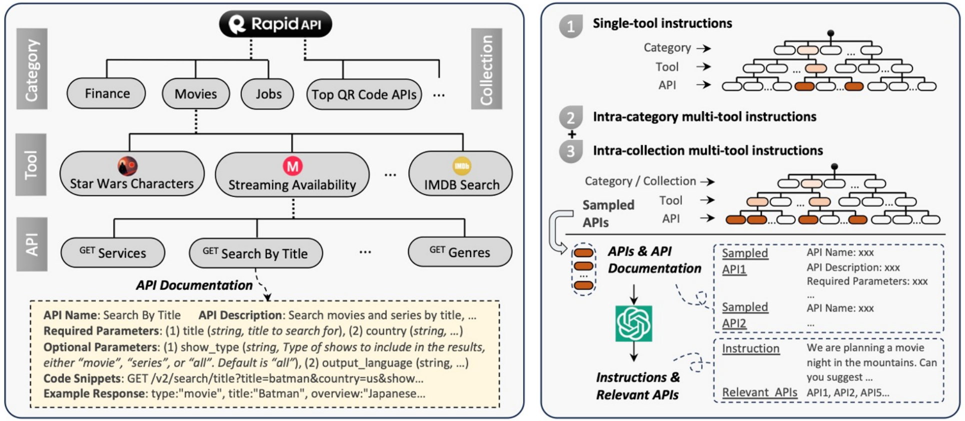


Figure 3: The hierarchy of RapidAPI (left) and the process of instruction generation (right).

Dataset Construction: Instruction Generation

$$\underset{\{\text{API}_1,\cdots,\text{API}_N\}\in\mathbb{S}_{\text{API}},\{\text{seed}_1,\cdots,\text{seed}_3\}\in\mathbb{S}_{\text{seed}}}{\text{ChatGPT}}(\{[\mathbb{S}_1^{\text{rel}},\text{Inst}_1],\cdots,[\mathbb{S}_{\text{N'}}^{\text{rel}},\text{Inst}_{\text{N}'}]\}|\text{API}_1,\cdots,\text{API}_{\text{N}},\text{seed}_1,\cdots,\text{seed}_3).$$

Set of APIs

Seed examples

(Relevant APIs,
Instructions)

Dataset Construction: Solution Path Annotation

- CoT is limited
  - Doesn’t explore total solution space
- Depth First Search Based Decision Tree
  - Can give up and explore a new node

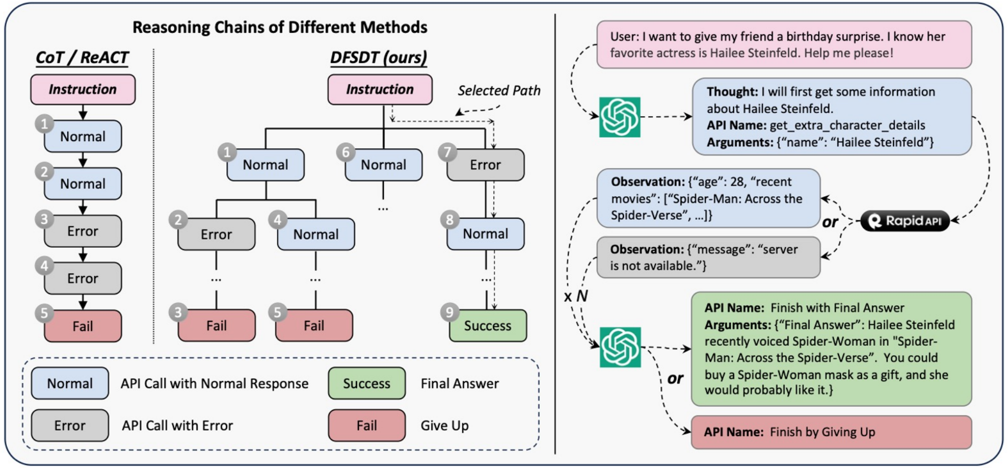


Figure 4: A comparison of our DFSDT and conventional CoT or ReACT during model reasoning (left). We show part of the solution path annotation process using ChatGPT (right).

Evaluation

What is a good model?

- Three Levels of Generalization (Focus on Unseen Situations):
  - Instruction Level:
    unseen instructions *for the same set of tools in the training data*
  - Tool:
    unseen tools *that belong to the* **same (seen) category** *of the tools in the training data*
  - Category Level:
    unseen tools *that belong to a* **different (unseen) category** *of tools in the training data*
- Three Types of Generalization Tasks (Focus on Task Complexity):
  - I1 (Single-tool instructions)
  - I2 (Intra-category multi-tool instructions)
  - I3 (Intra-collection multi-tool instructions)

Evaluation

Tools

- ToolEval:
  - Pass Rate: The percentage of tasks the model successfully completes within certain constraints (like limited API calls or time).
  - Win Rate: A comparison between two solution paths (e.g., solutions generated by different models). It measures which solution is better based on criteria such as accuracy, completeness, and reasoning. ChatGPT is used to evaluate and decide which solution is superior.
- NDCG (Normalized Discounted Cumulative Gain):
  - used to evaluate the performance of the API Retriever. NDCG measures how well the retriever ranks relevant APIs for a given task. It calculates how closely the retrieved APIs match the ground truth (the most appropriate APIs) by assigning a relevance score to the top results.

Main experiments

- ToolLLaMA Model:
  - Fine-tuned **LLaMA-2 7B model** using the ToolBench dataset.
  - Extended context length to **8192 tokens** to handle long API responses.
  - Evaluated on **three generalization levels**:
    - Instruction Level: Unseen instructions using the same tools.
    - Tool Level: Unseen tools in the same category.
    - Category Level: Unseen tools from different categories.
- Three Task Scenarios:
  - I1: Single-tool instructions.
  - I2: Intra-category multi-tool instructions.
  - I3: Intra-collection multi-tool instructions.

Results

Model	Method	I1-Inst.		I1-Tool		I1-Cat.		I2-Inst.		I2-Cat.		I3-Inst.		Average	
		Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win
ChatGPT	ReACT	41.5	-	44.0	-	44.5	-	42.5	-	46.5	-	22.0	-	40.2	-
	DFSDT	54.5	60.5	<u>65.0</u>	<u>62.0</u>	60.5	57.3	75.0	<u>72.0</u>	71.5	**64.8**	62.0	69.0	64.8	64.3
Claude-2	ReACT	5.5	31.0	3.5	27.8	5.5	33.8	6.0	35.0	6.0	31.5	14.0	47.5	6.8	34.4
	DFSDT	20.5	38.0	31.0	44.3	18.5	43.3	17.0	36.8	20.5	33.5	28.0	65.0	22.6	43.5
Text-Davinci-003	ReACT	12.0	28.5	20.0	35.3	20.0	31.0	8.5	29.8	14.5	29.8	24.0	45.0	16.5	33.2
	DFSDT	43.5	40.3	44.0	43.8	46.0	46.8	37.0	40.5	42.0	43.3	46.0	63.0	43.1	46.3
GPT4	ReACT	53.5	60.0	50.0	58.8	53.5	<u>63.5</u>	67.0	65.8	72.0	60.3	47.0	<u>78.0</u>	57.2	<u>64.4</u>
	DFSDT	<u>60.0</u>	**67.5**	**71.5**	**67.8**	**67.0**	**66.5**	<u>79.5</u>	**73.3**	**77.5**	<u>63.3</u>	**71.0**	**84.0**	**71.1**	**70.4**
Vicuna	ReACT & DFSDT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Alpaca	ReACT & DFSDT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ToolLLaMA	ReACT	25.0	45.0	29.0	42.0	33.0	47.5	30.5	50.8	31.5	41.8	25.0	55.0	29.0	47.0
	DFSDT	57.0	55.0	61.0	55.3	<u>62.0</u>	54.5	77.0	68.5	<u>77.0</u>	58.0	<u>66.0</u>	69.0	66.7	60.0
	DFSDT-Retriever	**64.0**	<u>62.3</u>	64.0	59.0	60.5	55.0	**81.5**	68.5	68.5	60.8	65.0	73.0	<u>67.3</u>	63.1

Table 4: Main experiments of ToolBench. Win rate is calculated by comparing each model with ChatGPT-ReACT. A win rate higher than 50% means the model performs better than ChatGPT-ReACT. Apart from ToolLLaMA-DFSDT-Retriever, all methods use the oracle API retriever (i.e., ground truth API).

Results

Model	Method	I1-Inst.		I1-Tool		I1-Cat.		I2-Inst.		I2-Cat.		I3-Inst.		Average	
		Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win
ChatGPT	ReACT	41.5	-	44.0	-	44.5	-	42.5	-	46.5	-	22.0	-	40.2	-
	DFSDT	54.5	60.5	65.0	62.0	60.5	57.3	75.0	72.0	71.5	**64.8**	62.0	69.0	64.8	64.3
Claude-2	ReACT	5.5	31.0	3.5	27.8	5.5	33.8	6.0	35.0	6.0	31.5	14.0	47.5	6.8	34.4
	DFSDT	20.5	38.0	31.0	44.3	18.5	43.3	17.0	36.8	20.5	33.5	28.0	65.0	22.6	43.5
Text-Davinci-003	ReACT	12.0	28.5	20.0	35.3	20.0	31.0	8.5	29.8	14.5	29.8	24.0	45.0	16.5	33.2
	DFSDT	43.5	40.3	44.0	43.8	46.0	46.8	37.0	40.5	42.0	43.3	46.0	63.0	43.1	46.3
GPT4	ReACT	53.5	60.0	50.0	58.8	53.5	63.5	67.0	65.8	72.0	60.3	47.0	78.0	57.2	64.4
	DFSDT	60.0	**67.5**	**71.5**	**67.8**	**67.0**	**66.5**	79.5	**73.3**	**77.5**	63.3	**71.0**	**84.0**	**71.1**	**70.4**
Vicuna	ReACT & DFSDT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Alpaca	ReACT & DFSDT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	ReACT	25.0	45.0	29.0	42.0	33.0	47.5	30.5	50.8	31.5	41.8	25.0	55.0	29.0	47.0
ToolLLaMA	DFSDT	57.0	55.0	61.0	55.3	62.0	54.5	77.0	68.5	77.0	58.0	66.0	69.0	66.7	60.0
	DFSDT-Retriever	**64.0**	62.3	64.0	59.0	60.5	55.0	**81.5**	68.5	68.5	60.8	65.0	73.0	67.3	63.1

Table 4: Main experiments of ToolBench. Win rate is calculated by comparing each model with ChatGPT-ReACT. A win rate higher than 50% means the model performs better than ChatGPT-ReACT. Apart from ToolLLaMA-DFSDT-Retriever, all methods use the oracle API retriever (i.e., ground truth API).

Results

Model	Method	I1-Inst.		I1-Tool		I1-Cat.		I2-Inst.		I2-Cat.		I3-Inst.		Average	
		Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win
ChatGPT	ReACT	41.5	-	44.0	-	44.5	-	42.5	-	46.5	-	22.0	-	40.2	-
	DFSDT	54.5	60.5	65.0	62.0	60.5	57.3	75.0	72.0	71.5	**64.8**	62.0	69.0	64.8	64.3
Claude-2	ReACT	5.5	31.0	3.5	27.8	5.5	33.8	6.0	35.0	6.0	31.5	14.0	47.5	6.8	34.4
	DFSDT	20.5	38.0	31.0	44.3	18.5	43.3	17.0	36.8	20.5	33.5	28.0	65.0	22.6	43.5
Text-Davinci-003	ReACT	12.0	28.5	20.0	35.3	20.0	31.0	8.5	29.8	14.5	29.8	24.0	45.0	16.5	33.2
	DFSDT	43.5	40.3	44.0	43.8	46.0	46.8	37.0	40.5	42.0	43.3	46.0	63.0	43.1	46.3
GPT4	ReACT	53.5	60.0	50.0	58.8	53.5	63.5	67.0	65.8	72.0	60.3	47.0	78.0	57.2	64.4
	DFSDT	60.0	**67.5**	**71.5**	**67.8**	**67.0**	**66.5**	79.5	**73.3**	**77.5**	63.3	**71.0**	**84.0**	**71.1**	**70.4**
Vicuna	ReACT & DFSDT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Alpaca	ReACT & DFSDT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	ReACT	25.0	45.0	29.0	42.0	33.0	47.5	30.5	50.8	31.5	41.8	25.0	55.0	29.0	47.0
ToolLLaMA	DFSDT	57.0	55.0	61.0	55.3	62.0	54.5	77.0	68.5	77.0	58.0	66.0	69.0	66.7	60.0
	DFSDT-Retriever	**64.0**	62.3	64.0	59.0	60.5	55.0	**81.5**	68.5	68.5	60.8	65.0	73.0	67.3	63.1

Table 4: Main experiments of ToolBench. Win rate is calculated by comparing each model with ChatGPT-ReACT. A win rate higher than 50% means the model performs better than ChatGPT-ReACT. Apart from ToolLLaMA-DFSDT-Retriever, all methods use the oracle API retriever (i.e., ground truth API).

Results

- **ToolLLaMA + DFSDT consistently outperforms ReACT**, achieving higher pass and win rates across both simple and complex tasks.

- **ToolLLaMA demonstrates strong generalization capabilities**, effectively handling new APIs and instructions, performing close to GPT-4.

- **DFSDT is superior to ReACT**, especially for complex reasoning tasks involving multiple tools.

Out-of-distribution generalization

- Objective:
  - Test ToolLLaMA’s generalization ability to unseen APIs using the APIBench dataset.
  - Compare ToolLLaMA with Gorilla (LLaMA-7B fine-tuned on APIBench).
- Two Retriever Settings:
  - ToolLLaMA + Our API Retriever (used in main experiments).
  - ToolLLaMA + Oracle Retriever (provides exact APIs for task completion).
- Gorilla Settings:
  - Zero-shot (ZS): No API prompts during training.
  - Retrieval-aware (RS): Retrieved APIs included in training prompts.

Results

Method	HuggingFace		TorchHub		TensorHub	
	Hallu. (↓)	AST (↑)	Hallu. (↓)	AST (↑)	Hallu. (↓)	AST (↑)
ToolLLaMA + Our Retriever	10.60	**16.77**	15.70	**51.16**	6.48	40.59
Gorilla-ZS + BM25	46.90	10.51	17.20	44.62	20.58	34.31
Gorilla-RS + BM25	**6.42**	15.71	**5.91**	50.00	**2.77**	**41.90**
ToolLLaMA + Oracle	8.66	88.80	14.12	85.88	7.44	88.62
Gorilla-ZS + Oracle	52.88	44.36	39.25	59.14	12.99	83.21
Gorilla-RS + Oracle	**6.97**	**89.27**	**6.99**	**93.01**	**2.04**	**94.16**

Table 5: OOD generalization experiments on APIBench. For the Gorilla entries, ZS / RS means that Gorilla was trained in a zero-shot / retrieval-aware setting on APIBench. We report hallucination rate and AST accuracy.

Results

Method	HuggingFace		TorchHub		TensorHub	
	Hallu. (↓)	AST (↑)	Hallu. (↓)	AST (↑)	Hallu. (↓)	AST (↑)
ToolLLaMA + Our Retriever	10.60	**16.77**	15.70	**51.16**	6.48	40.59
Gorilla-ZS + BM25	46.90	10.51	17.20	44.62	20.58	34.31
Gorilla-RS + BM25	**6.42**	15.71	**5.91**	50.00	**2.77**	**41.90**
ToolLLaMA + Oracle	8.66	88.80	14.12	85.88	7.44	88.62
Gorilla-ZS + Oracle	52.88	44.36	39.25	59.14	12.99	83.21
Gorilla-RS + Oracle	**6.97**	**89.27**	**6.99**	**93.01**	**2.04**	**94.16**

Table 5: OOD generalization experiments on APIBench. For the Gorilla entries, ZS / RS means that Gorilla was trained in a zero-shot / retrieval-aware setting on APIBench. We report hallucination rate and AST accuracy.

Results

- **ToolLLaMA** performs exceptionally well in **out-of-distribution (OOD)** generalization, handling unseen APIs from the **APIBench** dataset.

- When using the **trained API retriever**, ToolLLaMA consistently achieves **low hallucination rates** and strong **AST accuracy**, outperforming Gorilla in both zero-shot (ZS) and retrieval-aware (RS) settings.

- With the **oracle retriever** (which provides exact APIs), ToolLLaMA reaches near-perfect AST accuracy, demonstrating its ability to use APIs accurately when given the right context.

ART: Automatic multi-step reasoning and tool-use for large language models

Author: Bhargavi Paranjape et al.

https://arxiv.org/abs/2303.09014

1. Introduction

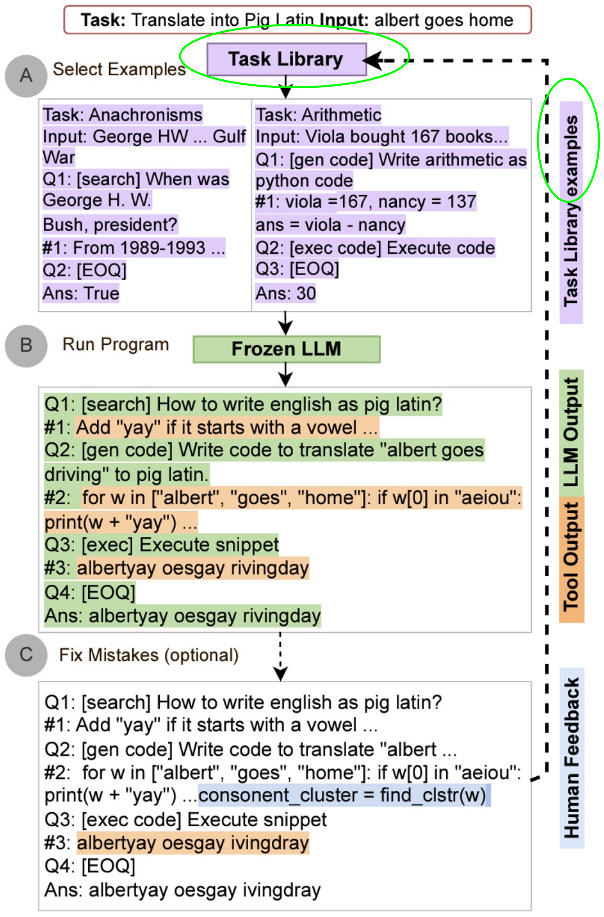
- Challenges with Current LLMs:
  - **Limited Multi-Step Reasoning:** LLMs struggle with tasks that require breaking down complex problems into smaller, intermediate steps.
  - **Lack of External Tool Use:** LLMs cannot naturally access external resources (e.g., search engines, calculators, code execution).

- Chain-of-Thought (CoT) Prompting:
  - requires **hand-crafted prompts** tailored to each task

1. Introduction

Overview of ART process:

- ART automatically breaks down complex tasks into smaller steps.
- It retrieves similar examples from a Task Library.
- Uses external tools from a Tool Library (e.g., search engines, code execution).
- Humans can provide feedback to refine or correct the process.



2. Technologies and Tools

- **Task:** the overall **problem** or objective to be solved.

- **Demonstration(**or **example): a step-by-step solution** to a specific task stored in **Task Library**
  - The input (the task or problem).
  - The step-by-step breakdown of how to solve the task (i.e., the program or series of actions to be taken).
  - The correct output (the answer or solution to the task).

- **Program:** the **set of instructions** that the LLM generates to solve a task.

2. Technologies and Tools

- **Large Language Models (LLMs):**
  - Uses **frozen LLMs** (e.g., GPT-3)
  - The model is **not retrained** but applied as-is to new tasks
- **Task Library:**
  - Collection of example tasks (**demonstrations**)
  - Uses a specific **Parsing Expression Grammar (PeG)** format to decompose tasks
- **Tool Library:**
  - **Search Engines**: For retrieving information like formulas or data.
  - **Code Execution Tools**: To perform calculations or run small programs to solve parts of a problem.

2.1 Task Library

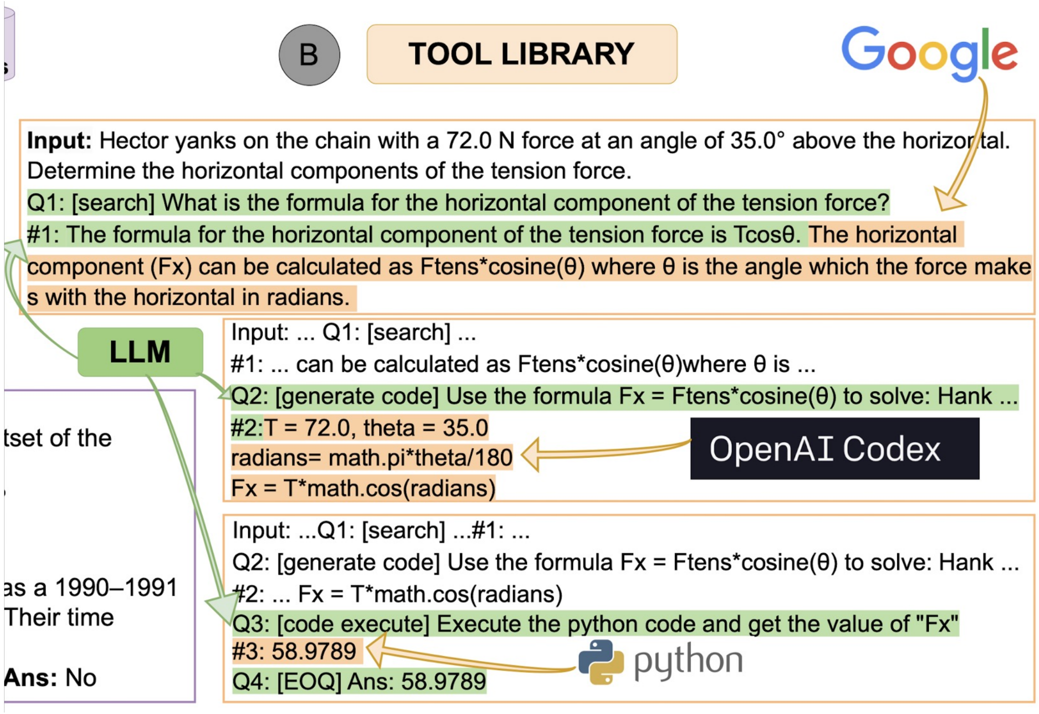
- **Big-Bench**: a collaborative benchmark that measures the capabilities and limitations of language models.
- Constructing the task library: group tasks in benchmark into the **5 tasks clusters**:
  - Arithmetic, Code, Search and question decomposition, Free-form reasoning, String Operations
  - Write programs for few instances in **2-4 tasks from each cluster**.
- **Program grammar**: a query language extends from the Decomposed Prompting Format
  - Represent decomposed reasoning steps sequentially
  - Incorporates function calls to external tools
  - A program: task input *node*(Input:...) + sub-step *nodes*(Qi:...,#i:...) + answer *node*(Ans:...)(Fig2)

2.2 Tool Library

Triggered when a sub-task query name matches a tool name in the task lib.

Seed the tool lib with the following tools(fig 2):

- Search: SerpAPI from Google search
- Code Generation: Codex model from OpenAI
- Code Execution: Virtual Python environment with pre-installed packages.



3. Intro to ART

3.1 Process of ART

1. **New Task Description**
   a. The model examines the task description to understand the problem.
2. **Retrieve Similar Tasks from the Task Library**
   a. ART searches the **Task Library** for similar examples.
   b. The examples (demonstrations) show how to break down similar tasks into smaller steps.
3. **Building the Prompt**
   a. ART uses retrieved examples to build a prompt that guides the LLM.
   b. The prompt shows the model how to decompose the new task into smaller steps, and may include instructions for using external tools.
4. **Program Generation**
   a. The LLM starts creating a step-by-step solution (program) for the task.
   b. Pauses generation whenever a tool is called and resuming generation after that.
   c. Tool Use: If the task requires external help (like searching or calculations), ART pauses, uses the tool, and integrates the result back into the program.
5. **Human Feedback (Optional)**
   a. Task library: add/correct decomposition demo
   b. Tool library: add more tool definitions and use examples

3.1 Process of ART

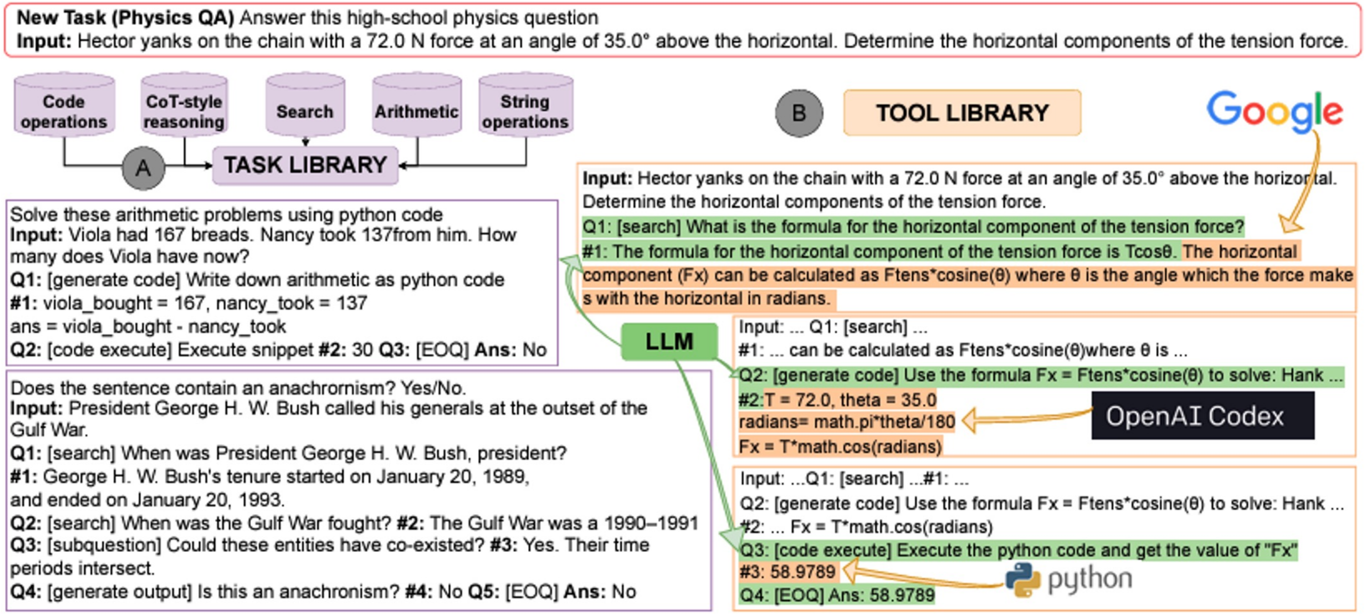


Figure 2: A run-through of ART on a new task, Physics QA. (A) Programs of related tasks like anachronisms and Math QA provide few-shot supervision to the LLM — related sub-steps and tools in these programs can be used by the LLM for cross-task generalization (shown in purple). (B) Tool use: Search is used to find the appropriate physics formula, and code generation and execution are used to substitute given values and compute the answer (shown in orange).

3.1 Process of ART - Human Feedback

Explore feedback in the form of debugging: use *edit* instead of create new programs from beginning:

- Correcting sub-step outputs
- Adding/removing sub-steps
- Adding calls to new tools(in the form of dictionary)
- …

3.1 Process of ART - Human Feedback

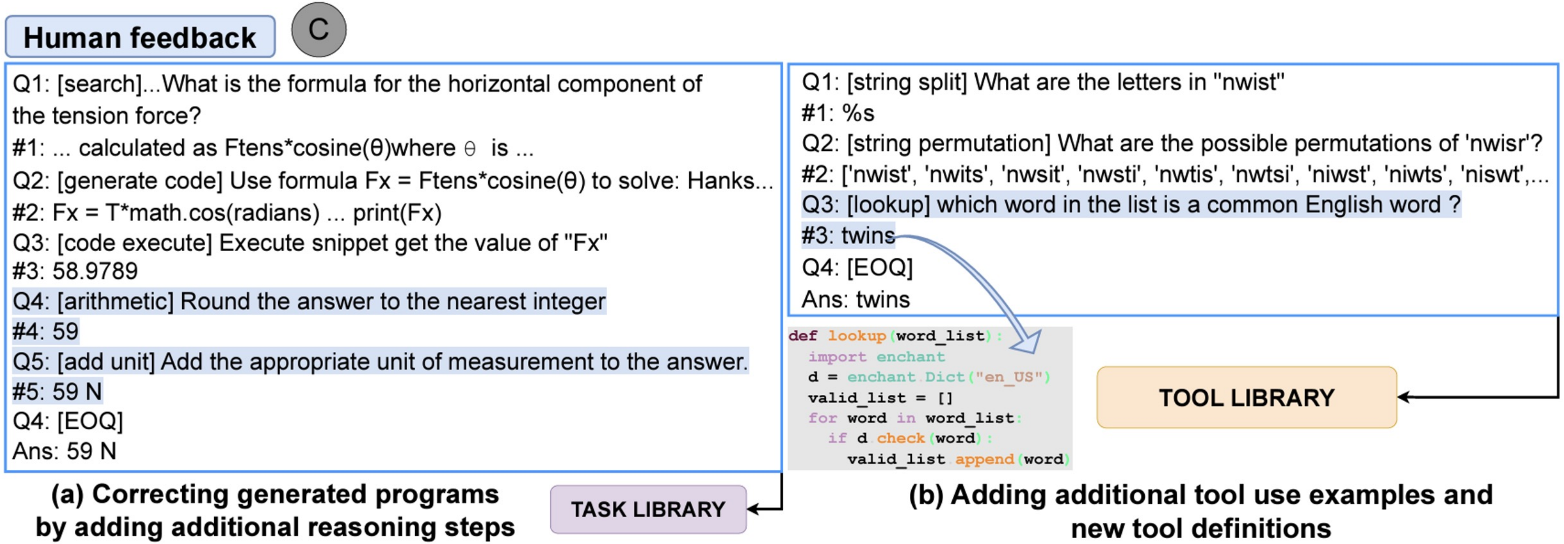


Figure 3: Human feedback to ART shown for (a) PQA where reasoning steps are added to the program and; (b) Word unscrambling where tool library is augmented with a new lookup tool.

Experiments

- Evaluation Datasets:
  - **15 tasks** from the ART **Task Library**.
  - 19 **new tasks** from **BigBench** for generalization testing.
  - Check cross-benchmark performance:**A subset of tasks** from **MMLU** benchmark
  - Compared ART to **Toolformer**, a model fine-tuned for tool use.
- Baseline models:
  - **Few-shot/Direct:** LLMs given examples of **input-output pairs** without reasoning steps.
  - **Auto-CoT:** Automatically generates **Chain-of-Thought** reasoning (step-by-step) but **without tool use**.
  - **ART-tool:** ART with **tool use turned off**, to check the gains from using tools.
  - **GPT-3 Best**: Best published results with **manual human supervision** and tool use.
- Experiment Setup:
  - Used **InstructGPT** (GPT-3) as the LLM and **Codex** for code generation/execution.
  - **Temperature** set to 0.3 (for focused response generation).
  - **3 seed tasks** and **2 demonstrations** per task.
  - Results averaged over **5 runs** to ensure consistency.

Results

- Task library:
  - Stronger than AutoCoT
  - Comparable to the best GPT-3 results
- Test tasks:
  - BigBench test tasks
  - Other Benchmarks

Task Name (Cluster)	Few Shot	AutoCot	ART w/o Tool Use	ART	GPT-3 Best
Anachronisms (Search)	71.3[5]	51.48	70.87	75.66	-
Musique (Search)	2.03[5]	12.88	10.04	19.19	15.2[3]
Hindu Knowledge (Search)	85.02 [5]	73.03	83.42	87.98	-
Known Unknown (Search)	68.90 [5]	56.09	80.43	80.43	-
Δ with ART (Search)	**+9.0**	**+17.44**	**+4.6**		+4.0
Elementary Math QA (Arithmetic)	56.40[7]	74.52	58.04	68.04	-
Aqua-rat (Arithmetic)	20.54[7]	34.41	36.29	54.20	54.1[4]
GSM8K (Arithmetic)	7.79[7]	21.99	53.4	71.00	71.6[4]
Navigate (Arithmetic)	60.7[7]	61.7	72.4	72.4	85.90[1]
Δ with ART (Arithmetic)	**+30.0**	**+18.25**	**+11.4**		-4.7
K'th letter concatenation (String)	3.2[5]	0.64	8.19	40.00	98.0[2]
Language games (String)	35.14[5]	18.58	11.19	23.08	-
Date Understanding (String)	37.53[5]	38.90	52.05	-	70.41[1]
Auto Debugging (Code)	62.94[5]	38.24	55.29	62.94	-
Code Description (Code)	97.99[7]	88.67	84.67	88.00	-
Formal Fallacies (CoT)	44.84[5]	56.4	64.76	-	58.4[1]
Hyperbation (CoT)	62.72[5]	55.4	80.80	-	72.4[1]
Δ with ART (Misc)	**+9.6**	**+16.4**	**+13.7**		-15.4
Δ with ART (Overall)	**+14.90**	**+17.17**	**+7.91**		-9.0

LLM+P: Empowering Large Language Models with Optimal Planning Proficiency

Author: Bo Liu et al.

https://arxiv.org/abs/2304.11477

WashU Medicine

1. Intro

Large language models:

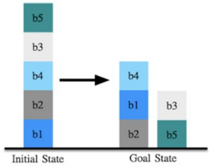
Good at *linguistic competence*: HOW to say;
Bad at *functional competence*: WHAT to say.

- Easily fooled by asking for the solution to a problem that requires knowledge about how the world works.

- Objective: enable LLMs to solve planning problems correctly without altering the LLMs themselves.

A Failure Example of GPT-4 in Planning

Problem (P1): You have 5 blocks. One cannot place more than one block on another block. b5 is on top of b3. b4 is on top of b2. b2 is on top of b1. b3 is on top of b4. b1 is on the table. b5 is clear. Your arm is empty.
Your goal is to move the blocks.
b1 should be on top of b2.
b3 should be on top of b5.
b4 should be on top of b1.



GPT-4:

```
1. Move b5 from b3 to the table.
2. Move b3 from b4 to b5.
3. Move b4 from b2 to the table.
4. Move b2 from b1 to b5.
5. Move b1 from the table to b2.
6. Move b4 from the table to b1.
```

2. Preliminary

2.1 The *Classical Planning Problem*

A planning problem **P** is defined by a tuple $\langle \mathscr{S}, s^{init}, \mathscr{S}^{G}, \mathscr{A}, f \rangle$

$\mathscr{S}$: A finite and discrete set of states

$s \in \mathscr{S}$: Defined by the values of a fixed set of variables. $s^{init} \in \mathscr{S}$

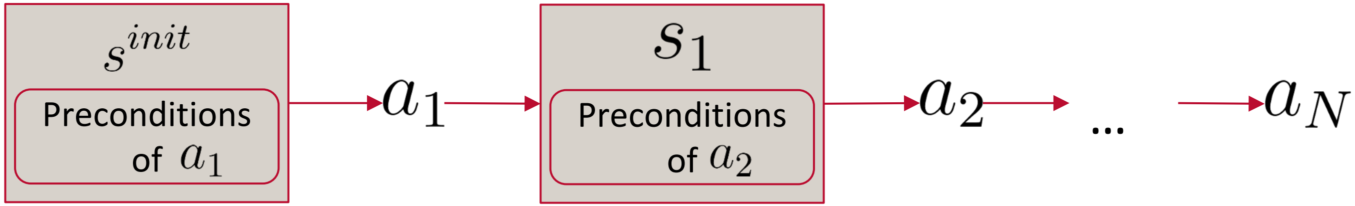
$\mathscr{S}^{G} \subset \mathscr{S}$: A set of goal states which are usually specified as a list of goal conditions

$\mathscr{A}$: A set of symbolic actions

f : The underlying state transition function. Takes the current state and action as input and outputs the corresponding next state

2.1 The *Classical Planning Problem*

π : The solution to P in the form of $\langle a_1, a_2, \ldots, a_N \rangle$



2.2 *Planning Domain Definition Language* (PDDL)

PDDL serves as a standardized encoding of classical planning problems.

1. Domain PDDL file: Provides a lifted representation of the underlying rules of the world. It includes a set of predicates that define the state space $\mathscr{S}$ and the actions with their preconditions and effects.

2. Problem PDDL file: Provides a list of objects to ground the domain, the problem's initial state s^{init} and goal conditions $\mathscr{S}^{G}$.

3. Related Work

A. Classical Planning:

- Have been widely used in robot systems;
- Use PDDL or answer ser programming(ASP) as the action language;
- Guaranteed to be logically correct;
- Able to find Optimal plans.

B. Planning with Large Language Models:

- Success in extracting task knowledge and decompose commands or instructions for robots in natural language;
- Lack of long-horizon reasoning ability: Ineffective plan;
- Need iteratively querying LLMs.

3. Related Work

C. Augmenting LLMs with External Modules:

- Web knowledge
- Human-in-the-loop
- Retrieval-augmented language modeling paradigm
- Calculators(What we are using)

Similar work: LLMs+PDDL+SayCan (dataset): SayCan has a limited scope which leads that models achieved high success rate. Also lack domain PDDL file leads to infeasible plans.

LLM+P: Does not rely on fine-tuning or re-training of LLMs.

4. Method

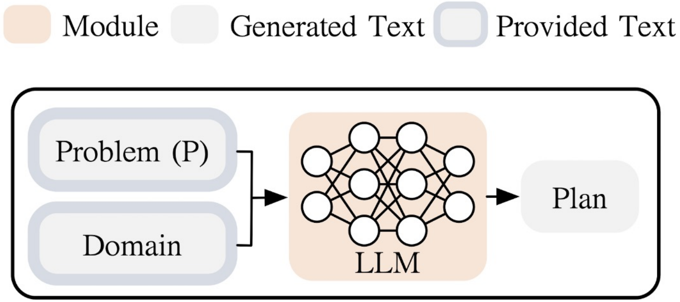
A. LLM as a PDDL Writer

Intuition:

View PDDL as a different language than English.

Example:

The generated file appears to have the correct PDDL syntax but uses a made-up predicate (empty) and misses the initial condition that b1 is on the table.



LLM-As-Planner

An Example PDDL Problem File Written by GPT-4

Prompt: Description of Problem (P1) + Provide me with the problem PDDL file that describes the planning problem directly without further explanations.

GPT-4 (the generated problem PDDL):

```
(:objects b1 b2 b3 b4 b5 - block)
(:init (on b5 b3) (on b4 b2) (on b2
b1) (on b3 b4) (clear b5) (empty))
(:goal (and (on b1 b2) (on b3 b5)
(on b4 b1)))
```

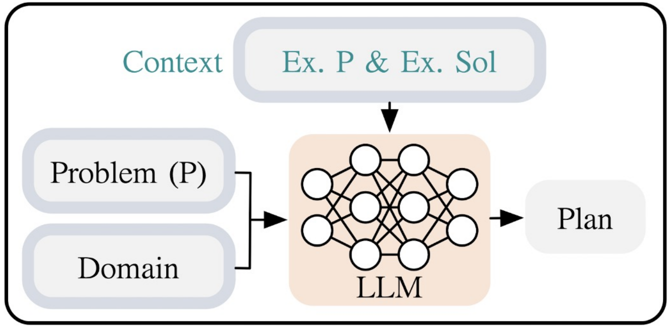
4. Method

B. In-Context Learning:

LLMs' ability to perform unseen downstream tasks by simply conditioning on a few input-label pairs (demonstrations).

Example:

When the context is included with the prompt from the example above, the resulting PDDL problem file is directly solvable by the planner.



LLM-As-Planner (In-context Learning)

An Example PDDL Problem File Written by GPT-4 with Context

Context (shaded):

An example planning problem is: You have 5 blocks. b2 is on top of b5. b5 is on top of b1. b1 is on top of b4. b3 is on top of b2. b4 is on the table. b3 is clear. Your arm is empty. Your goal is to move the blocks. b4 should be on top of b3. The problem PDDL is:

```
(:objects b1 b2 b3 b4 b5)
(:init (arm-empty) (on b1 b4) (on
b2 b5) (on b3 b2)
        (on-table b4) (on b5 b1)
(clear b3))
(:goal (and (on b4 b3))))
```

Prompt: Description of Problem (P1) + Context + Provide me with the problem PDDL file that describes the planning problem directly without further explanations.

GPT-4 (the generated problem PDDL):

```
(:objects b1 b2 b3 b4 b5 )
(:init (arm-empty) (on-table b1)
(on b2 b1) (on b3 b4)
        (on b4 b2) (on b5 b3) (clear
b5)))
(:goal (and (on b1 b2) (on b3 b5)
(on b4 b1)))
```

Planner:

```
(unstack b5 b3) (putdown b5)
(unstack b3 b4) (stack b3 b5)
(unstack b4 b2) (putdown b4)
(unstack b2 b1) (putdown b2)
(pickup b1) (stack b1 b2) (pickup
b4) (stack b4 b1)
```

4. Method

C. LLM + Classical Planner (LLM + P)

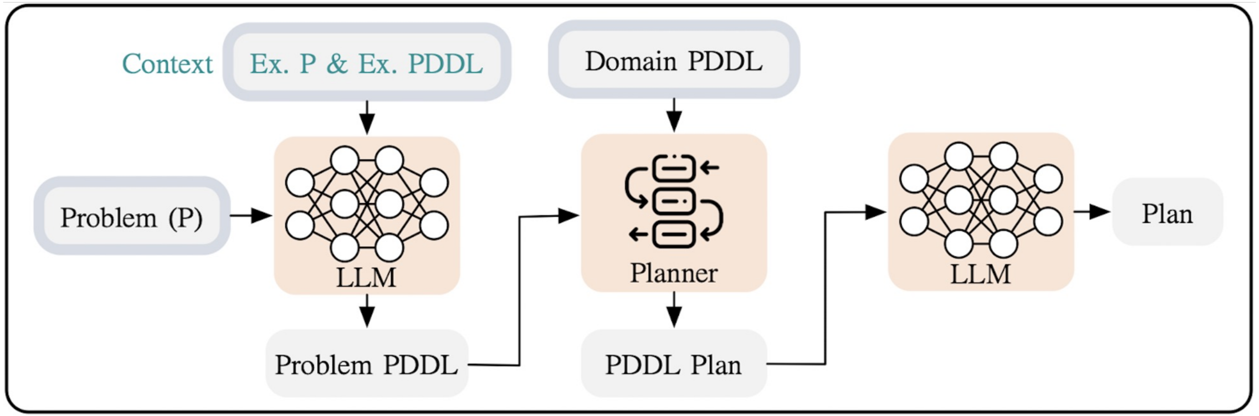
When posed a natural language description of a planning problem, we hope the LLM:

1) Outputs a problem description suitable as input to a general-purpose planner;

2) Solves the problem using the general-purpose planner;

3) Converts the output of the planner back to natural language (or connects to action executors of a robot).

4. Method



LLM + P (In-context Learning)

1. Agent is provided with a minimal example;
2. Provided with a new problem P;
3. The LLM then uses the in-context learning to infer the problem PDDL file corresponding to P;
4. Feed the problem PDDL file into planner and generate a PDDL plan;
5. LLM translate the PDDL plan back into natural language.

4. Method

The **assumptions** we need for LLM+P are:

1. A robot knows **when** to trigger LLM+P based on its conversation with a human user;

2. A domain PDDL is provided to **define the actions** that the robot is capable of. This specification is task- agnostic — the entities relevant to the task are specified in the LLM-generated problem PDDL;

3. A simple **problem description** in natural language and its corresponding **problem PDDL file** are also provided.

5. Experiments

1. How well does LLM-AS-P work? To what extent can state-of-the-art LLMs and LLM-based reasoning methods be directly used for planning? (**Not at all**)

2. How does LLM+P work compare to LLM-AS-P? (**Much better**)

3. What role does the context play in the success of LLM+P? (**It's crucial**)

4. Can LLM+P help make service robots more efficient on realistic tasks? (**Yes**)

5. Experiments 1

Domain	Success Rate %				
	LLM^-	LLM	LLM^{ToT}	$LLM+P^-$	LLM+P
BARMAN	0	0	0	0	**20 (100)**
BLOCKSWORLD	20	15 (30)	0 (5)	0	**90**
FLOORTILE	0	0	0	0	**0**
GRIPPERS	25 (60)	35 (50)	10 (20)	0	**95 (100)**
STORAGE	0	0 (25)	0	0	**85**
TERMES	0	0	0	0	**20**
TYREWORLD	5	15	0	0	**10 (90)**

TABLE I: Success rate % of applying LLM-AS-P without context (LLM^-), LLM-AS-P (LLM), Tree of Thoughts (LLM^{ToT}), LLM+P without context (LLM^-), and LLM+P.

5. Experiments 1

LLM-AS-P:

- Infeasible plans;
- Lack of inference ability: Completely failed at domain with complex spatial relationship;
- Bad at long-horizon problems: Ignores the requirements, times out, cannot keep track of properties.

LLM+P:

- Most failed cases are due to mis-specified problem files;
- Context is important.

5. Experiments 2

Robot Demonstration:

LLM-AS-P outputs a sub-optimal plan which takes the *bottle* to the *pantry* first and travels back for the *soup can*, with a total cost of 31.

Tidy-Up Problem PDDL Generated by LLM+P

Problem (P): You are a home robot with one gripper. The distance between coffee table and side table is 10. The distance between coffee table and pantry is 20... You are at the coffee table. There is a mustard bottle... Your goal is to move objects to their destinations...

Problem PDDL generated by LLM+P:

```
(:objects coffee-table side-table
recycle-bin pantry - location
mustard-bottle soup-can - object)
(:init (= (total-cost) 0) (=
(distance coffee-table side-table)
10) (= (distance coffee-table
pantry) 20) ... (robot-at
coffee-table) (at mustard-bottle
coffee-table) (at soup-can
side-table) (hand-empty) )
(:goal (and (at mustard-bottle
pantry) (at soup-can recycle-bin)))
(:metric minimize (total-cost)) )
```

6. Conclusion

1. Combine LLMs with Classical Planners;
2. Define a diverse set of benchmark problems;
3. Experiment on these benchmark problems;
4. Use home robot to solve manipulation task (in natural language);
5. Limitation: can’t recognize which prompt is suitable for LLM+P.

6. Conclusion

1. A good general methodology: leverage classical planners to empower large language models with optimal planning capabilities.

2. Key: Focus LLMs on translating.

Limitation/Future work:

- When a prompt should be processed by LLM+P;
- Enabling the LLM to auto-detect when and how to apply LLM+P;
- Reducing LLM+P’s dependency on information by humans, potentially involving fine tuning.

Future Directions for LLM as Agents

Author: Lei Wang et al.

https://arxiv.org/pdf/2308.11432

WashU Medicine

Future Directions for LLM as Agents

- Enhancing Role-playing Capability:
    - *Agents need to convincingly simulate **specialized roles** (e.g., programmer, chemist), which requires more **fine-tuning** with specific role data.*
- Generalized Human Alignment:
    - *Aligning agents with **human values**, both positive and negative, is crucial for accurate real-world simulations.*
- Improving Prompt Robustness:
    - *LLMs need more **resilient prompts** to ensure consistent behavior across complex multi-step tasks, like **planning** or **memory-based operations**.*
- Mitigating Hallucination:
    - *Reducing **LLM-generated false information** (hallucinations) is critical, especially in tasks requiring high precision (e.g., code generation, security).*
- Managing Knowledge Boundaries:
    - *Limit LLM's vast **pre-existing knowledge** to avoid overly informed agent behavior and improve realistic user simulations.*
- Boosting Efficiency:
    - *Enhancing the **speed of LLM responses** is essential to ensure agents perform real-time tasks effectively.*

Thank you!