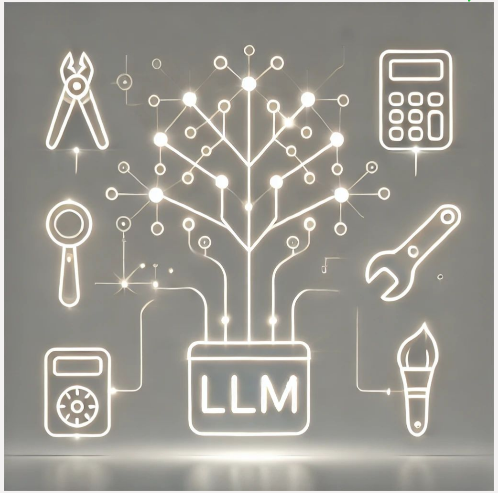
LLMs In The Imaginarium: Tool Learning Through Simulated Trial and Error

Rishitha Kalicheti
Elena Kote

Motivation

We want LLMs to interact with the world through tools



How accurately do LLMs use tools for which they have been trained?

- Existing LLMs: 30% to 60% correctness rate
- Existing methods focus on broad coverage of tools and flexibility of using new tools
  - **ICL:** flexible but lower accuracy
  - **Fine-tuning:** mainly focus tools not seen on training; catastrophic forgetting

- Authors propose **STE (Simulated Trial and Error)**
  - Help LLMs master tools they have been trained on

Related Work

Tool-augmented LLMs

Models with retrieval/search engines

Fine-tuning

ICL

More diverse tools: program executors, translation, APIs

Learning from Feedback

LLMs can improve their predictions with feedback

Self-reflection can be used as a form of feedback for progressive learning

Models with dynamic memory

Growing memory of user and environmental feedback

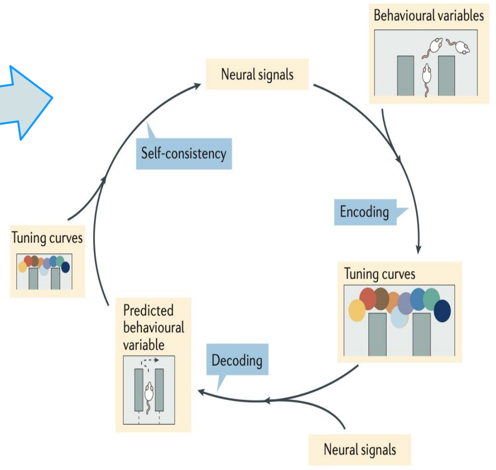
Fine-grained short term memory

Distilled long-term memory

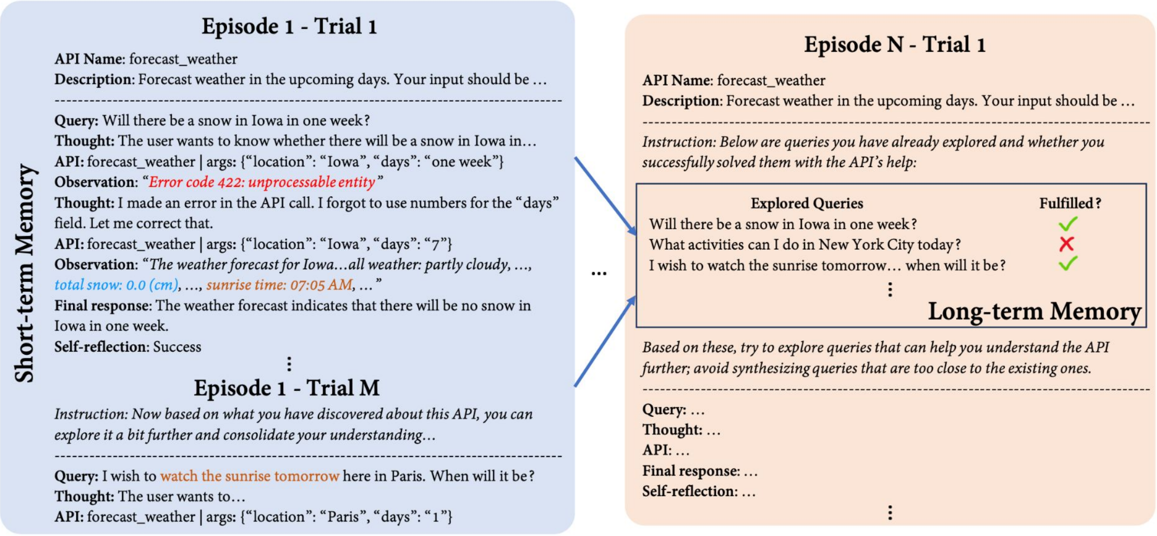
Simulated Trial and Error (STE)

A biologically-inspired method for tool-augmented LLMs.

- Inspired by biological precedents (humans, apes, rats)
  - Trial and error
  - Imagine possible scenarios
  - Remember previous experiences
- STE steps:
  - LLM simulates possible tool use cases
  - LLM iteratively interacts with tool
  - Reflect on current trial
  - Explore using short and long-term memory



1. Exploration



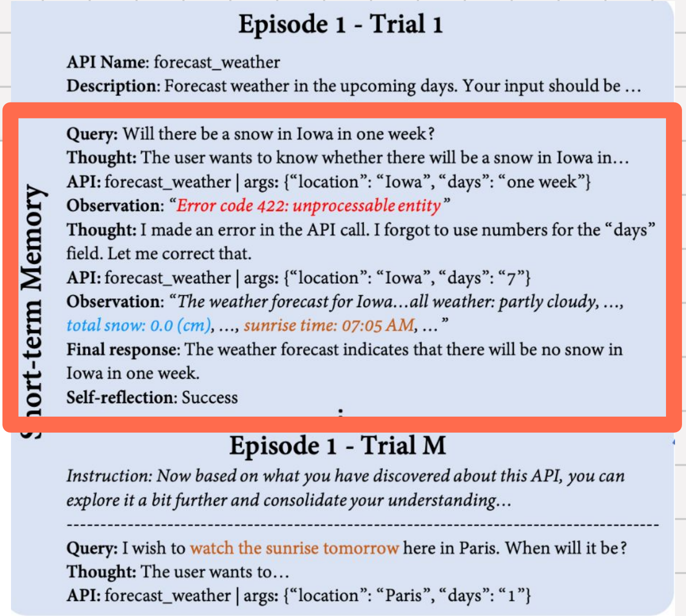
- LLM interacts with API to gain as much information from it as possible, on a series of Episodes and Trials

- **Episodes**: Long-term memory
- **Trials**: Short term memory

- On each Trial
    - Imagine plausible query
    - Fulfill query by using API
    - Reflect on trial

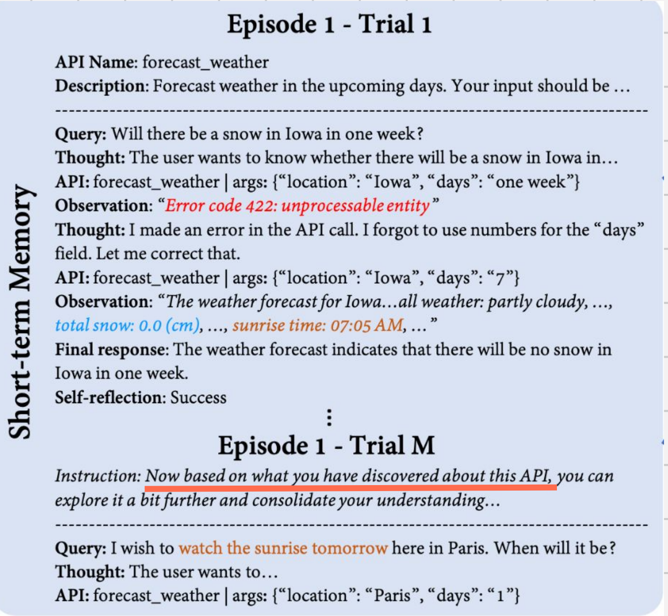
Iterative self-refine with execution feedback

- LLM learns from execution feedback to refine its own outputs
- ReAct format
  - Verbalize internal thoughts
  - Make an action (API call)
  - Observe execution feedback
  - Repeat
    - until adequate API response (or max #calls reached)

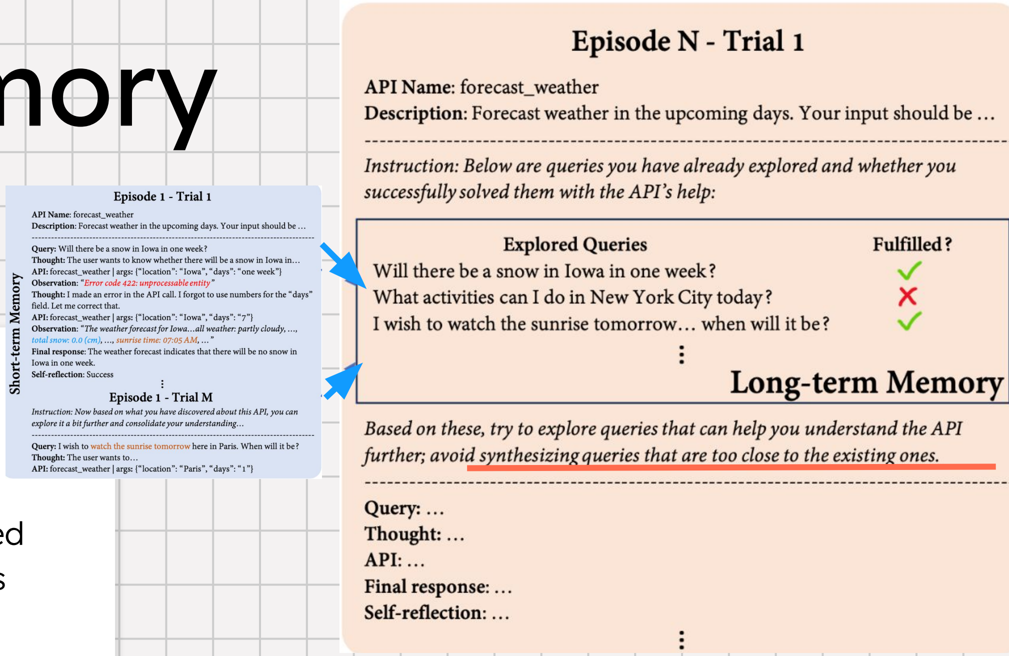


Short - term memory

- Each episode starts with a fresh short-term memory
  - **Only remember recent trials**
- Learns from recent fine-grained successes and failures
  - **Syntactic/semantic errors**
- Explores the API in greater depth
  - **Based on previous observations of API**



Long - term memory



- Store distilled trial-and-error experiences from previous experiences
- Keep track of past-explored queries and if fulfilled
- Helps LLM continually expand exploration across different episodes
  - **Instructed to imagine scenarios distant to existing ones**

2. Exploitation

- Trials obtained from exploration stage utilized for
  - ICL
  - Fine-tuning
- For each trial
  - Extract synthesized user query
  - LLM's last API call
  - Execution results
  - Final response from trial trajectory
- **Filter** with GPT-4
  - Judge validity of each example
- **Paraphrase** valid examples for each API

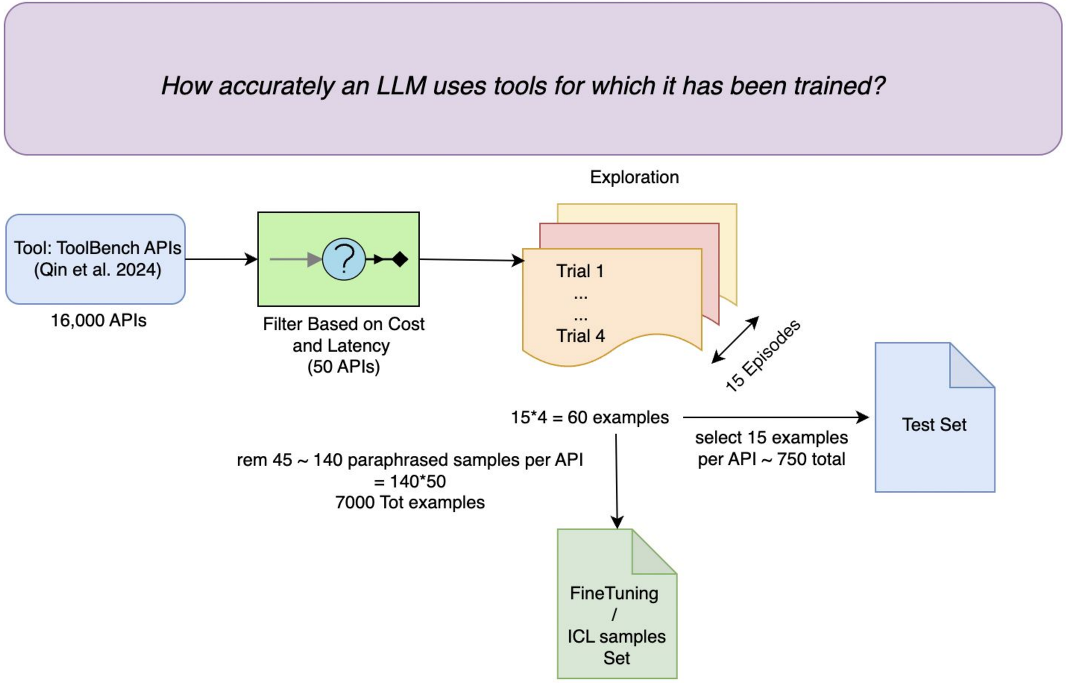
An assistant is trying to respond to the user query with the help of some APIs. The APIs that the assistant has access to are as follows:
{api_descriptions}
Now, your task is to evaluate how well the assistant did the job. Check carefully the following aspects of the assistant's response:
1) whether the response answers the user's query in an informative way. For example, if the API calls are unsuccessful and the agent can't find the answer to the request, you should say "No."
2) whether the response is faithful with respect to the execution results of the API calls. The response should not include information that cannot be supported by the API call feedback,
3) whether the assistant used the API calls appropriately. For example, the assistant should always use relevant API calls for queries about up-to-date information or complex calculations,
For each of the three aspects, you should say "Yes" or "No" indicating whether the assistant did a good job in that aspect, and explain the reason behind your judgment. Your output should follow the format below, where "<explanation>" should be your actual explanation for the corresponding judgment:
1) Yes/No. <explanation>
2) Yes/No. <explanation>
3) Yes/No. <explanation>
Now, the user query is:
{query}
The assistant's API calls and the corresponding execution results are:
{chains}
The assistant's final response is:
{final_ans}
Now, your evaluation is (remember to follow the previous format):

Table 12: Prompt for example filtering.

Below you will be given a user query. Try to paraphrase it in a different way while preserving its meaning. The query is:
{query}
Your paraphrase of the query:
=========
Can you try to paraphrase it again in a new way? Avoid coming up with something too close to your previous ones. Your paraphrase:

Table 13: Prompt for query paraphrasing.

Experimental setup



- Filtered and Selected 50 Tool APIs that are free, less latency and not rate limited from ToolBench
- Used ChatGPT (16k) for exploration /paraphrasing.
- Used GPT-4 (8k) for final filtering.
- Finally generated Test Set and Fine-Tuning Dataset.
- Next slide, ICL and Fine-Tuning setup

Dataset Example

[ ] add_date
- { } 0
  - query : "What is the date 90 days from 2022-10-14?"
  - [ ] API_name_list
  - api_descriptions : "API_name: add_date Description: Add days to a date. Date should be p
  - action : "add_date"
  - action_input : "{ "date": "2022-10-14", "days": 90 }"
- { } 1
  - query : "Can I calculate the date that is 14 days after 2022-01-01?"
  - [ ] API_name_list
  - api_descriptions : "API_name: add_date Description: Add days to a date. Date should be p
  - action : "add_date"
  - action_input : "{"date": "2022-01-01", "days": 14}"
- { } 2
  - query : "How can I calculate the date that is 20 days before 2022-01-01?"
  - [ ] API_name_list
  - api_descriptions : "API_name: add_date Description: Add days to a date. Date should be p
  - action : "add_date"
  - action_input : "{ "date": "2022-01-01", "days": -20 }"
- { } 3
  - query : "What is the date 50 days before a specific date 2022-01-01?"
  - [ ] API_name_list
  - api_descriptions : "API_name: add_date Description: Add days to a date. Date should be p
  - action : "add_date"
  - action_input : "{ "date": "2022-01-01", "days": -50 }"

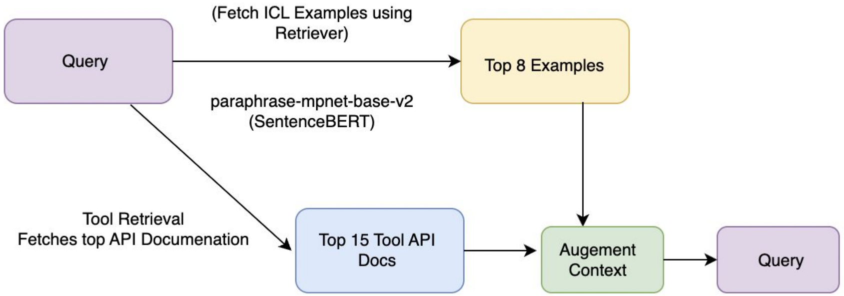
[ ] forecast_weather
- { } 0
  - query : "What will be the weather forecast for the next 10 days in Tokyo?"
  - [ ] API_name_list
  - api_descriptions : "API_name: forecast_weather Description: Forecast weather ir
  - action : "forecast_weather"
  - action_input : "{ "location": "Tokyo, JP", "days": 10 }"
- { } 1
  - query : "What is the weather forecast for the next 3 days in Rome, Italy?"
  - [ ] API_name_list
    - 0 : "forecast_weather"
  - api_descriptions : "API_name: forecast_weather Description: Forecast weather ir
  - action : "forecast_weather"
  - action_input : "{ "location": "Rome, IT", "days": 3 }"
- { } 2
  - query : "What is the weather forecast for the next 3 days in Tokyo?"
  - [ ] API_name_list
  - api_descriptions : "API_name: forecast_weather Description: Forecast weather ir
  - action : "forecast_weather"
  - action_input : "{ "location": "Tokyo", "days": 3 }"

Experimental setup

Baselines:

- **ToolLLaMA-v2** - Llama-2-7B and fine-tuned on 126K tool use examples synthesized by ChatGPT- 3.5-turbo for general tool-use
- Models: **Llama-2-Chat-7B/13B**, **Mistral-Instruct-7B** (Both FT, ICL). **GPT-3.5-turbo/GPT-4 (ICL Only)**

ICL Setup:



Fine-Tuning Setup:

- LLMs fine-tuned on STE do not need such API documentation in the context
- Reduces the inference cost - Why?
- Cause of less input tokens.

Evaluation Metrics

Correctness

Measures the accuracy of both:

1. Predicted API name
2. Arguments provided

Method of Validation:

1. Strict Value Ranges: **String matching** on arguments (Eg: Int, float check)

2. **ChatGPT judge**s the correctness on natural lang args

Wellformedness

Percentage of examples with valid API calls

1. No syntax errors.
2. No formatting errors.

API Match

Percentage of examples where the model selects the ground-truth API.

API Call matching.

Evaluation Metrics - Examples

Failure Cases

Correctness

Ground Truth:

translateText(input="How are you?", targetLanguage="French")

Model Prediction:

translateText(input="What's up?", targetLanguage="French")

Argument input value mismatch, judged incorrect

Wellformedness

Model Prediction:

getWeather[location="Paris", date="2024-11-17"]

Invalid syntax getWeather() should be the right call

API Match

User Query:

"What will the weather be in San Francisco tomorrow?"

Ground Truth API:

getWeather(city="San Francisco", date="2024-11-17")

Predicted API:

searchCityInfo(city="San Francisco")

Results

Setting	Base Model	Wellformed?	API Match	Correctness
Baseline	ToolLLaMA-v2	98.1	49.0	37.3
	Llama-2-Chat-7B	34.5	40.2	10.7
	Llama-2-Chat-13B	79.3	53.6	32.7
	Mistral-Instruct-7B	61.7	69.3	30.1
	GPT-3.5-turbo (16k-0613)	96.9	77.6	60.5
	GPT-4 (8k-0613)	96.1	78.1	60.8
ICL w/ STE	Llama-2-Chat-7B	58.3	86.7	41.7
	Llama-2-Chat-13B	87.5	86.6	62.9
	Mistral-Instruct-7B	69.9	88.4	47.9
	GPT-3.5-turbo (16k-0613)	97.6	90.8	75.6
	GPT-4 (8k-0613)	97.7	92.8	76.3
Fine-tuning w/ STE	Llama-2-Chat-7B	**99.2**	94.9	73.3
	Llama-2-Chat-13B	98.9	95.1	74.3
	Mistral-Instruct-7B	99.1	**95.8**	**76.8**

1. **STE** is effective with both ICL and fine-tuning.
2. ICL with STE shows significant improvement over baselines (correctness metric).
3. Fine-tuning with STE significantly outperforms ICL.
4. This is likely because fine- tuning allows injecting a much wider range of tool use examples into a model than ICL.
5. This shows the importance of pre-incorporating tool-use knowledge into LLMs, i.e., the importance of exploration step to fetch FT examples/dataset.

GPT-4 Error Analysis (Before -> After ICL with STE)

User query: Which parks in San Francisco have hiking trails?
Ground truth API name: `search_places`
Description for ground truth API: Run Places search. Your input should be a json (args json schema): {"query": string, }
Ground truth arguments: {"query": "parks in San Francisco with hiking trails"}

Predicted API: `Geographic coordinates by placename`
Description for predicted API: Returns geographic coordinates for the given placename (city, village, etc.). The method returns the place whose name is most similar to the search string.", "required_parameters": [{"name": "name", "type": "STRING", "description": "Placename", "default": "London"}, {"name": "lang", "type": "ENUM", "description": "Two-letter language code (ISO639-1). The following values are available: en (english), ru (russian)", "default": ""}], "optional_parameters": [{"name": "country", "type": "STRING", "description": "Two-letter country code, ISO-3166 (optional). Default is all countries.", "default": ""}]
Predicted Arguments: {"name": "San Francisco", "lang": "en"}

Wrong API call
36.7% -> 19.0%

STE helps better illustrate fine-grained semantics

User query: What are the geographic coordinates for the city of Sydney, Canada?
Ground truth API name: Geographic coordinates by placename
Description for ground truth API: Returns geographic coordinates for the given placename (city, village, etc.). The method returns the place whose name is most similar to the search string. Required parameters: [{"name": "name", "type": "STRING", "description": "Placename", "default": "London"}, {"name": "lang", "type": "ENUM", "description": "Two-letter language code (ISO639-1). The following values are available: en (english), ru (russian)", "default": ""}]. Optional parameters: [{"name": "country", "type": "STRING", "description": "Two-letter country code, ISO-3166 (optional). Default is all countries.", "default": ""}]
Ground truth arguments: `{"name": "Sydney", "country": "CA", "lang": "en"}`

Predicted API: Geographic coordinates by placename
Description for predicted API: Returns geographic coordinates for the given placename (city, village, etc.). The method returns the place whose name is most similar to the search string.", "required_parameters": [{"name": "name", "type": "STRING", "description": "Placename", "default": "London"}, {"name": "lang", "type": "ENUM", "description": "Two-letter language code (ISO639-1). The following values are available: en (english), ru (russian)", "default": ""}], "optional_parameters": [{"name": "country", "type": "STRING", "description": "Two-letter country code, ISO-3166 (optional). Default is all countries.", "default": ""}]
Predicted Arguments: `[{"name": "Sydney", "country": "CA"}`

Missing/Wrong API arguments
26.7% -> 10.0%

User query: What are the main causes of climate change?
Ground truth API name: `search`
Description for ground truth API: The input is an exact entity name. The action will search this entity name on Wikipedia and returns the first five sentences if it exists. If not, it will return some related entities to search next. Your input should be a json (args json schema): {"entity": string, }
Ground truth arguments: {"entity": "causes of climate change"}

Predicted API: `search_general`
Description for predicted API: Run query through GoogleSearch and parse result. Your input should be a json (args json schema): {"query": string, }
Predicted Arguments: {"query": "main causes of climate change"}

Table 6: Error example for GPT-4 (category: hard to judge correctness). Here GPT-4 calls the Google Search API which could also be a valid choice besides the ground truth.

Hard to evaluate examples
36.7% -> 16.7%

Some tools have overlapping functionalities/are time-sensitive

Mistral-Instruct-7B Error Analysis (FT with STE)

Commonsense/world knowledge (47.7%)

=> Scale or more knowledge retrieval

Language Understanding (31.6%)

=> Use stronger base LLM

Grounding (21.1%)

=> fuzzy-matching; add decoding constraints

User query: What is the current advisory information for the Union City station?

Gold API call: BART - Advisory information
Args: {"cmd": "bsa", "orig": "UCTY"}

- ***Description***: *The BART API gives you access to pretty much all of the BART service and station data available… Required parameters: [{"name": "cmd", "type": "STRING", … name": "orig", "type": "STRING", "description": "Optional station filter. Uses 4 character BART station abbreviations…}]*

Predicted API call: BART - Advisory information
Args: {"cmd": "bsa", "orig": "UNION"}

(a)

User query: What are the ongoing giveaways for PC game keys on the GOG platform?

Gold API call: GamerPower - Filter & Group Giveaways
Args: {"platform": "gog", "type": "game.key"}

- ***Description***: *"Find all free games, loot and giveaways with this giveaway tracker API powered by GamerPower.com! Access programmatically the best giveaways in… Required parameters: [{"name" : "platform", "type": "STRING"}, {"name": "type", "type": "STRING", "description": …}]*

Predicted API call: GamerPower - Filter & Group Giveaways
Args: {"platform": "pc", "type": "game.key"}

(b)

User query: Can you provide the total number of matches played between Leeds United and Sheffield Wednesday in the English Premier League?

Gold API call: Football Dolphin - Head to head statistics
Args: {"first_team" : "Leeds", "second_team": "Sheffield Weds", "type_of_statistics": …}

- ***Description***: *This API returns statistical data about English Premier League… Required parameters: [{"name": "first_team", type: "STRING", description: Enter first team from all available teams: Arsenal, …, Ipswich, Leeds, Leicester, Liverpool, …}, {"name": …}]*

Predicted API call: Football Dolphin - Head to head statistics
Args: {"first_team": "Leeds United", "second_team": "Sheffield Weds", "type_of_statistics": …}

(c)

Ablation Study on Llama-2-Chat-7B

Setting	Wellformed?	API Match	Correctness
Full STE	99.2	94.9	73.3
– Exec. feedback	89.9	79.4	50.5
– Short. Mem.	99.7	70.6	53.9
– Long. Mem.	98.7	79.9	59.7
– Self-reflection	99.3	81.7	60.1

Execution Feedback: (-) Increase in ill-formed API calls (e.g., incorrect syntax or formatting)

Feedback ensures alignment with syntax/formatting requirements.

Short-Term Mem. (-) Misses details and fails to adapt based on prior errors. (Drop in correctness)

Misses exploration into fine-grained attributes (e.g., humidity, UV index, sunrise time). No specificity.

Short-Term Mem. Allows learning from immediate execution results. (i.e., previous trials in same episode)

Setting	Wellformed?	API Match	Correctness
Full STE	99.2	94.9	73.3
– Exec. feedback	89.9	79.4	50.5
– Short. Mem.	99.7	70.6	53.9
– Long. Mem.	98.7	79.9	59.7
– Self-reflection	99.3	81.7	60.1

Ablation Study on Llama-2-Chat-7B

Setting	Wellformed?	API Match	Correctness
Full STE	99.2	94.9	73.3
– Exec. feedback	89.9	79.4	50.5
– Short. Mem.	99.7	70.6	53.9
– Long. Mem.	98.7	79.9	59.7
– Self-reflection	99.3	81.7	60.1

Long-Term Mem. (-) Trials become repetitive and uninformative. (Drop in Correctness)

Long-Term Memory: Maintains diverse queries (e.g., locations, attributes, time).

Both Long and short-term Mems. significantly boost specificity, diversity, and correctness in trial-and-error learning

Self-Reflection

Crucial for maintaining informative long-term memory.

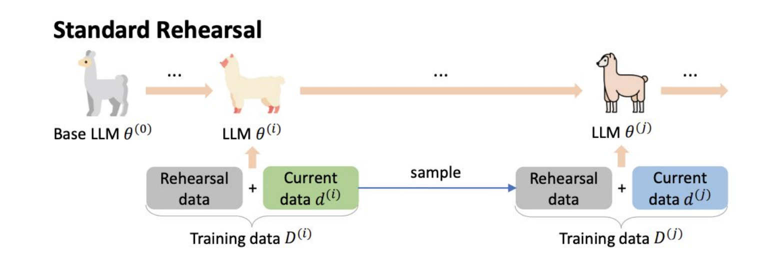
Enables the LLM to refine exploration and improve trial outcomes.

Setting	Wellformed?	API Match	Correctness
Full STE	99.2	94.9	73.3
– Exec. feedback	89.9	79.4	50.5
– Short. Mem.	99.7	70.6	53.9
– Long. Mem.	98.7	79.9	59.7
– Self-reflection	99.3	81.7	60.1

Continual Tool Learning

Catastrophic Forgetting of Fine-Tuned LLMs -> No Flexibility as new Tools get added.

Continual Learning with simple Rehearsal Strategy.



Huang, Jianheng, et al. "Mitigating catastrophic forgetting in large language models with self-synthesized rehearsal." *arXiv preprint arXiv:2403.01244* (2024).

Experimental Setup:
Tools are split into 4 batches to simulate incremental learning.

Rehearsal Strategy:

During each training round:

- Add 10% of previous tool-use examples (prev batch) into a **replay buffer**.
- Include 2,000 random examples from FlanV2 for general non-tool-use capabilities.

Evaluation Metrics/Benchmarks:

- Correctness score: Performance on previously learned tools after learning new tools.
- General Capabilities: MMLU, BBH

Model Used: Llama-Flan

Continual Tool Learning - Results

	Batch 1	Batch 2	Batch 3	Batch 4	All APIs	MMLU	BBH
Llama-Flan	-	-	-	-	-	37.2	39.5
CL-Round 1	80.6	-	-	-	-	39.6	36.8
CL-Round 2	1.7 → 76.1	87.7 → 84.1	-	-	-	40.2	38.9
CL-Round 3	0.0 → 70.6	56.9 → 84.1	68.9 → 65.6	-	-	39.2	37.5
CL-Round 4	0.0 → 65.0	38.5 → 88.7	25.0 → 66.1	71.8→ 70.3	34.7 → 72.8	38.5	39.1
Llama-FT	73.3	87.2	68.3	67.2	74.1	38.7	40.8

Rehearsal enables flexible addition of new tools without **catastrophic forgetting**.

Effective with STE.

Without Rehearsal:

Drastic forgetting of previously learned tools.
Tools learned in earlier batches are most severely impacted.

With Rehearsal:

Forgetfulness Mitigated: Retains tool-use capabilities comparable to fine-tuned models (Llama-FT).
General Language Abilities Preserved: Performs well on MMLU and BBH.

Limitations

Iterative Improvement

Current Exploration-Exploitation setup over relies on strong models

Ex. GPT4 for train tool query generation (paraphrasing + filtering).

Prior works on Iterative learning using feedback loops and RL.

Authors say that eventually as the **weaker models become more capable** (after STE Fine-tuned) -> Can close the gap

Planning

Another orthogonal direction is about Tool use and comprehensive planning.

Planning multiple tool calls to fulfill complex queries (like **multihop queries**)

If planning is successful we don't need FT dataset anymore, could just align models.

Beyond Context limit & Potential overfitting

Use **Retrieval models** or Hierarchal compressed memory models to solve context limit issue.

It's difficult to teach the model **when *not* to use a tool**.

To solve this, we can use negative examples (e.g., using contrastive objectives)

Tool unlearning?

The problem of unlearning is important as tools could get **deprecated/outdated**

ToolkenGPT allows plug-and-play adaptation while enabling learning with large-scale examples

CREATOR: Tool Creation for Disentangling Abstract and Concrete Reasoning for Language Models

- LLM ability limited by API availability and ability
- CREATOR:
    - LLMs create their own tools using documentation and code realization
    - Disentangles **abstract** tool creation and **concrete** decision execution
- Outperforms current CoT, PoT and tool-using baselines

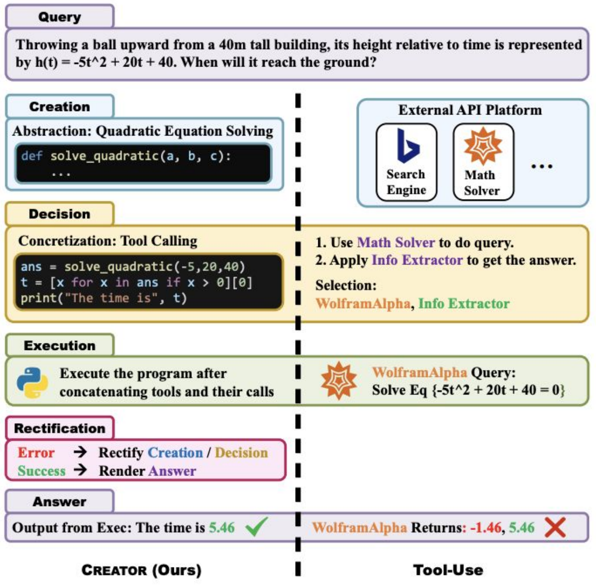


Figure 1: The difference between CREATOR and a general tool-using framework.

Thank You!