Washington University in St. Louis
James McKelvey School of Engineering

CSE561 Presentation: Language Models as Agents

Presenters:

- *Deyuan Yang*
- *Yancheng Jin*
- *Mingrui Ye*

Toolformer: Language Models Can Teach Themselves to Use Tools

Timo Schick et al., Meta AI (2023)

Presentation by Deyuan Yang

Teaching models to use tools, not just scale parameters.

The fundamental Problem with LLMs

Strength

- Creative Writing
- Conversation
- Reasoning patterns

Weakness

- Calculation errors
- Outdated facts
- Temporal confusion

Factual inaccuracy and hallucinations

Poor Mathematical reasoning

Limited multilingual capability

Existing Solutions and Their Limitations

Human Supervision Approach	Task-specific Approach
• High Cost: massive annotations • Human bias • Limited scale	• Not generalizable • requires retraining

Gap: No general, self-supervised approach to tool learning

Toolformer’s Core Innovation

Self-supervised Tool Learning

- No human annotations
- Maintain generality
- Learns when and how to use tools

- **Key**: Let the model decide what's useful using its own predictions and teaches itself how to use external APIs
- **Self-Supervise**: Use perplexity reduction as training signal
- **General Approach**: Works with any tool that has text-based API and Maintain core language modeling ability

Example

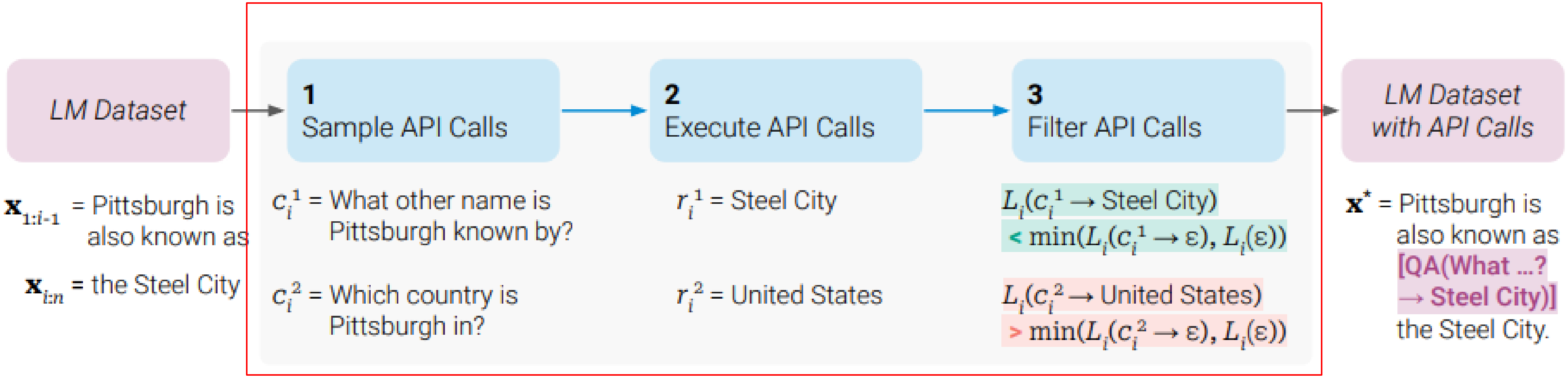
The New England Journal of Medicine is a registered trademark of [QA(“Who is the publisher of The New England Journal of Medicine?”) → Massachusetts Medical Society] the MMS.

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.

The Three-Step Learning Process



Step 1. **Sampling**: Generate potential API calls using in-context learning

Step 2. **Execution**: Actually call the APIs to get real results

Step 3. **Filtering**: Keep only calls that reduce future token prediction loss

Output: Augmented dataset with helpful API calls

Fine-tune the model on augmented dataset

Model learns when and how to use tools

Technical Deep Dive: Sampling API Calls

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:

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

- **In-Context Learning**: Provide few-shot examples of API usage
- **Position Sampling**: Compute probability of starting API call at each position
- **Call Generation**: Sample actual APU calls given the context
- **Example**: The Nile has length <API> QA ('Nile length') -> 6853km</API>6853 km

Technical Deep Dive: Smart Filtering

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

Weighted cross entropy loss

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

Loss when model sees API call and result

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

Minimum loss between no call or call without result

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

Decision criteria (filtering threshold): only keep calls that reduce loss significantly

keep calls that provide genuinely useful information

APIs and Tools

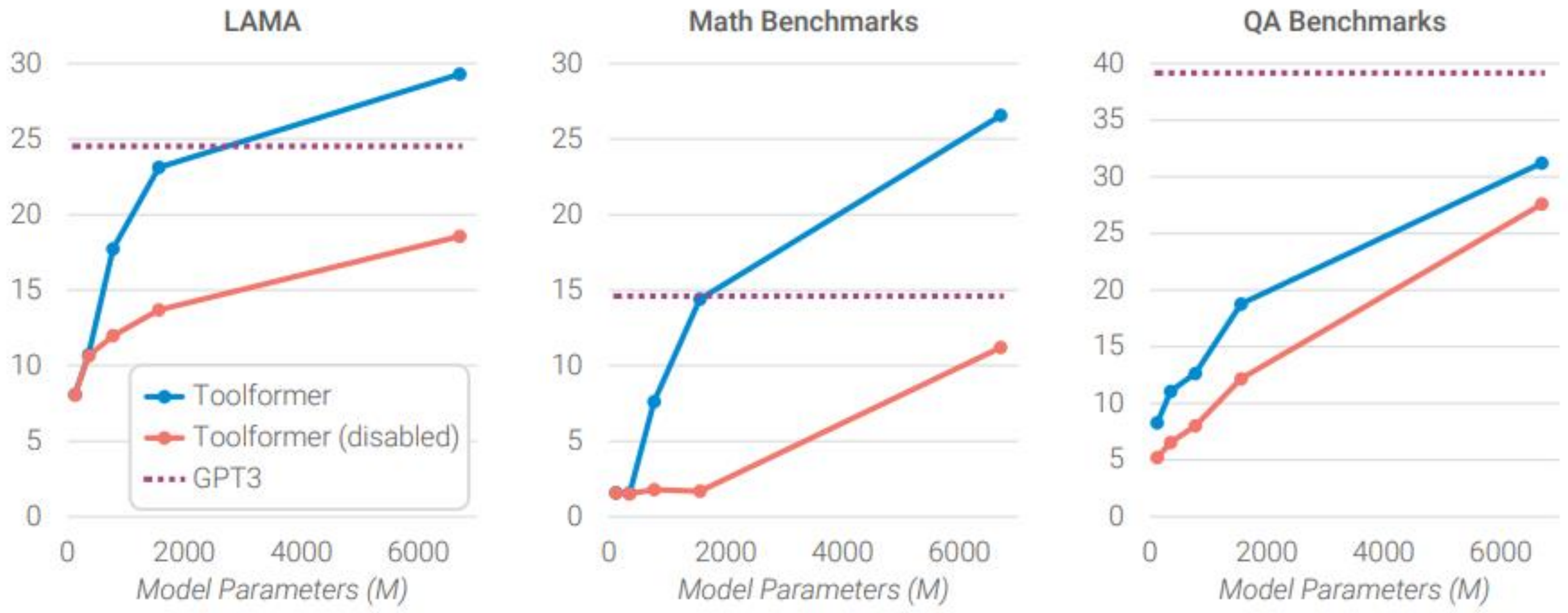
Tools	Purpose
Calculator	Arithmetic operations
QA System	Factual Questions (Atlas model)
Wikipedia Search	Information retrieval (BM25)
Machine Translation	200 languages (NLLB)
Calendar	Temporal Context

- Each tool addresses specific LLM weaknesses
- Only requirement: Text-based inputs and outputs

Experimental Setup

- **Base model**: GPT-J (6.7B parameters)
- **Dataset**: CCNet subset
- **Baselines (Comparison models)**: GPT-J, GPT-3 (175B), OPT (66B)
- **Tasks**: LAMA, Math, QA, Multilingual QA, Temporal reasoning
- **Evaluation**: zero-shot across multiple benchmarks

Key Result: Outperforming Giants



LAMA (factual): Toolformer: 33.8 vs GPT-3: 26.8

Math(SVAMP): Toolformer: 29.4 vs GPT-3: 10.0

Temporal (Dataset): Toolformer: 27.3 vs GPT-3: 0.8

Use appropriate tools for each task type

Tools Usage Analysis

Math Tasks	97.9% calculator usage
Factual Tasks	98.1% QA system usage
Multilingual	60%-95% translation usage
Temporal	54.8% calendar usage

- Model learns appropriate tool selection automatically
- High usage rates indicate reliable tool invocation
- Different tools dominate different task types

Critical Analysis and Ablations

Key Findings

- **No Generality Loss**: Perplexity unchanged (10.3 vs 10.3)
- **Emergent Ability**: Needs ~775M + parameters
- **Decoding Strategy**: k=10 works best for tool invocation

Summary

- Language modeling ability preserved
- Tool use emerges only at sufficient scale
- Inference strategy affects tool usage rates
- Performance gap remains between tool use vs no tool use

Limitations and Future Work

No Tool Chains	Cannot combine multiple tools
Not Interactive	Single-shot API calls only
Sample Inefficient	Many examples needed for rare tools
Prompt Sensitivity	Affected by input wording

- Current limitations provide clear research directions
- Future work: Tool chains, interactive use, iterative training
- Integration with reasoning frameworks like Chain-of-Thought

Conclusion and Impact

Old way: Scale parameters → New capabilities

New way: Augment with tools → Diminishing returns

- **Key Contribution**: Self-supervised tool learning framework
- **Impact**: Small models can outperform much larger ones
- Enhances zero-shot performance without extra data
- **Research Direction**: Augmentation over scaling

References

Schick, T., Dwivedi-Yu, J., Dessi, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). *Toolformer: Language Models Can Teach Themselves to Use Tools*. arXiv:2302.04761

Wang, B., & Komatsuzaki, A. (2021). *GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model*

Brown, T., et al. (2020). *Language Models are Few-Shot Learners*. NeurIPS

Wenzek, G., et al. (2020). *CCNet: Extracting High Quality Monolingual Datasets from Web Crawl Data*

Schick, T., & Schütze, H. (2021). *Generating Datasets with Pretrained Language Models*

TOOLLLM: FACILITATING LARGE LANGUAGE MODELS TO MASTER 16000+ REAL-WORLD APIS

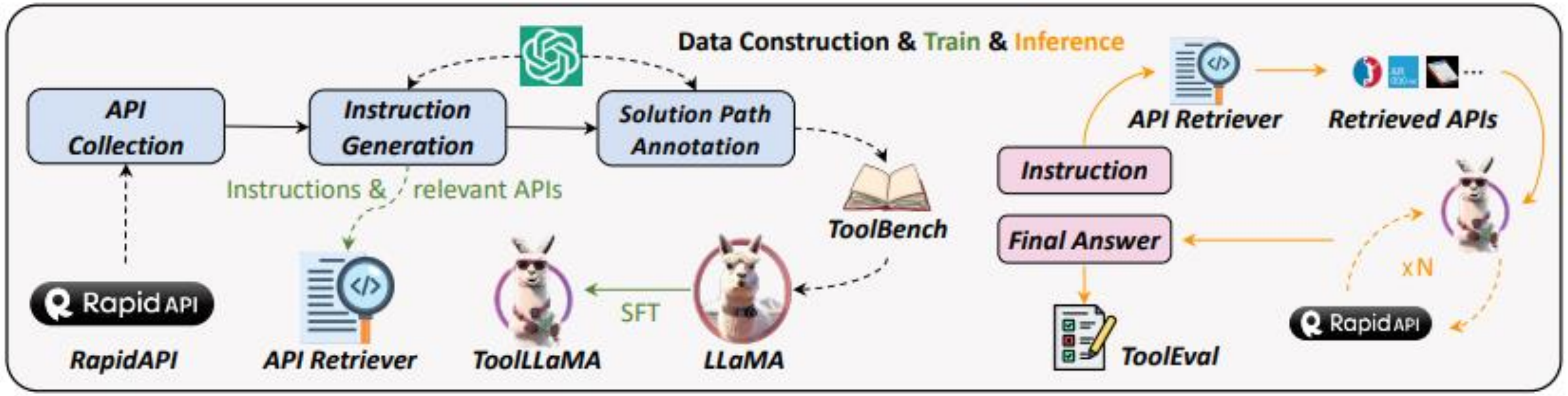
Yujia Qin et al.(2023)

Presentation by Yancheng Jin

Motivation

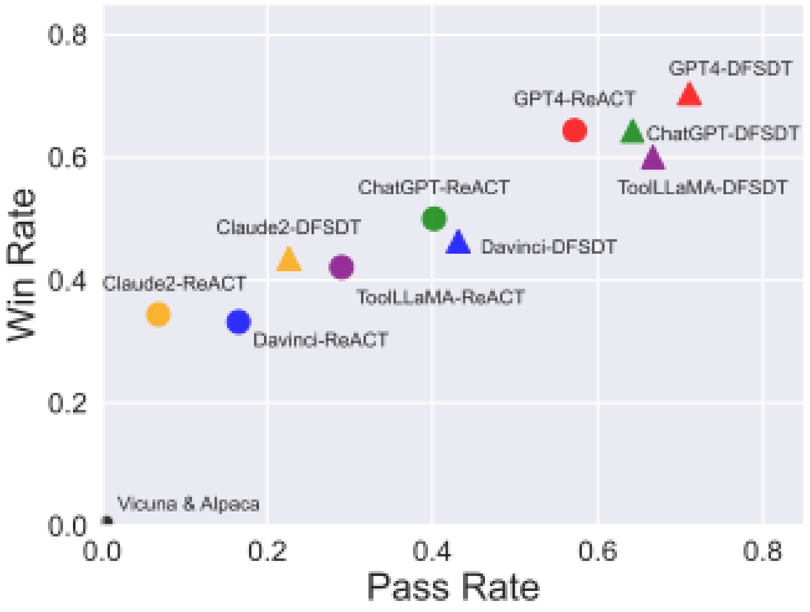
Goal: Why tool-use matters for LLMs

- Gap: open LLMs struggle with real API use vs. OpenAI closed models
- Real tasks need API selection, parameterization, sequencing
- Research question: How can we train open LLMs to master thousands of real APIs?



Key Gaps (Past Paper)

- Limited APIs/Realism: Few or no real REST APIs; small, low-diversity tool sets → weak generalization.
- Simplified Scenarios: Mostly single-tool, single-round; often assume users pre-select the “right” APIs (not scalable).
- Weak Planning/Reasoning: CoT/ReAct struggle on complex, long-horizon tasks.
- No Real Execution: Some don’t run APIs, missing response feedback critical for iterative planning.



ToolLLM

Def: A General Framework for Tool-Use in Open LLMs

Dataset (ToolBench):

API Collection: 16,464 real REST APIs from RapidAPI across 49 categories

Instruction Generation: ChatGPT creates single-tool + multi-tool instructions

Solution Path Annotation:

Evaluator (ToolEval):

Model (ToolLLaMA): LLaMA fine-tuned on ToolBench + Neural API Retriever

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**

Dataset-API Collection

RapidAPI Hub & Taxonomy

- Leading API marketplace, 49 coarse-grained categories. 500+ fine-grained collections

Hierarchy & Metadata Crawling

- A tool with API, name/desc, HTTP method, required/optional params, request body, executable code snippets, example responses

Quality Filtering

- Initial: 10,853 tools / 53,190 APIs=> Rigorous filtering =>3,451 tools / 16,464 APIs

Instruction Generation

Design Focus

- Diversity: Cover a wide range of API-use scenarios → better generalization & robustness
- Multi-tool Usage: Reflect real tasks requiring interleaved, multi-round tool execution

Generation Pipeline (Sample APIs → Generate Instructions)

- Define full API set S api; Sample a subset S(sub N)
- Prompt ChatGPT to understand APIs in S(sub N) and produce feasible instruction(Inst_*)
- Produce relevant API sets S*real ⊆ S(sub N) for each instruction

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

Prompt Composition

- High-level description of the instruction-generation task
- Comprehensive docs for each API (function, params, examples)
- Three in-context seed examples (separate seed pools for single-tool / multi-tool)

Sampling Strategies for Single Tool

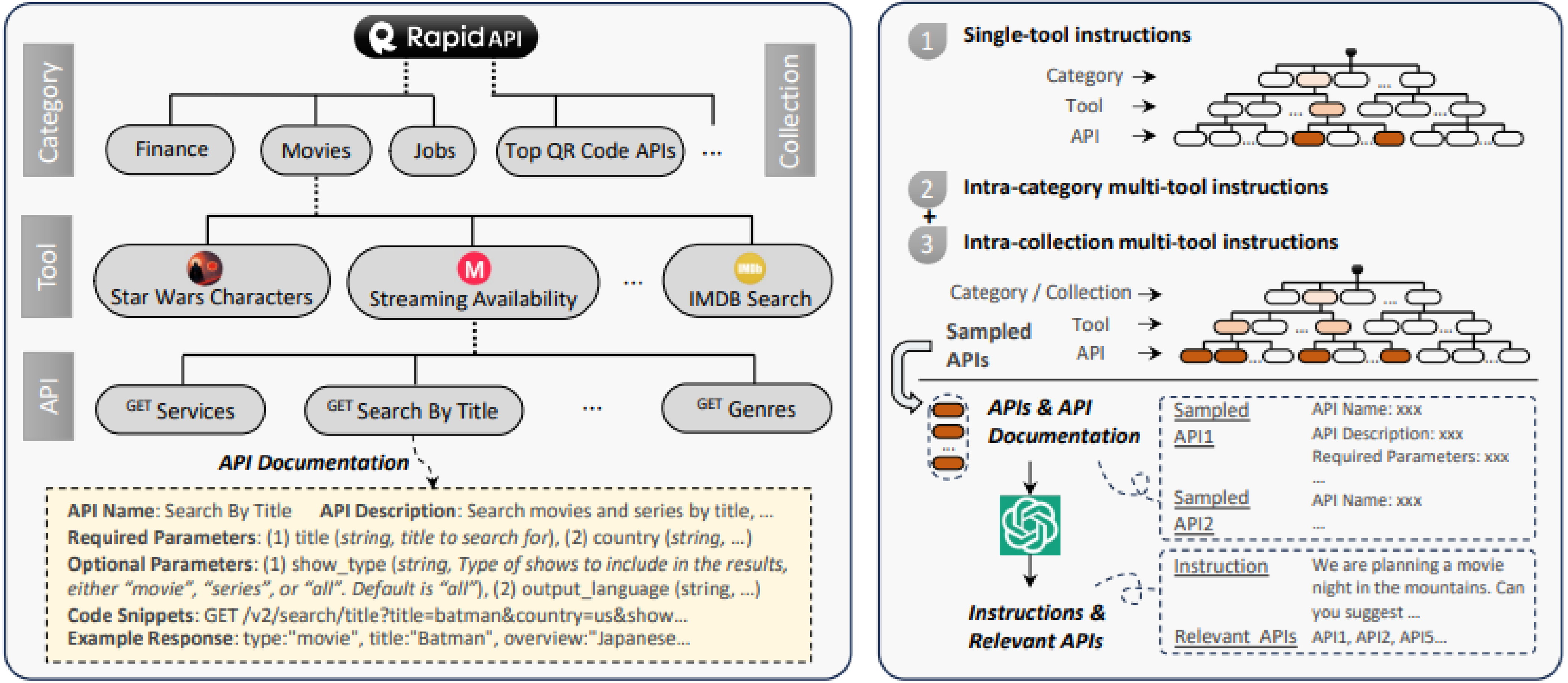


Figure 3: The hierarchy of RapidAPI (left) and the process of instruction generation (right).

Sampling Strategies for Multi-tool Setting

Why Specialized?

- sparse interconnections → random combinations yield irrelevant tool sets

Leverage RapidAPI Hierarchy for Multi-Tool

- I2: Intra-Category
- I3: Intra-Collection
- Rationale: tools within the same category/collection share functionality/goals → more coherent multi-tool workflows

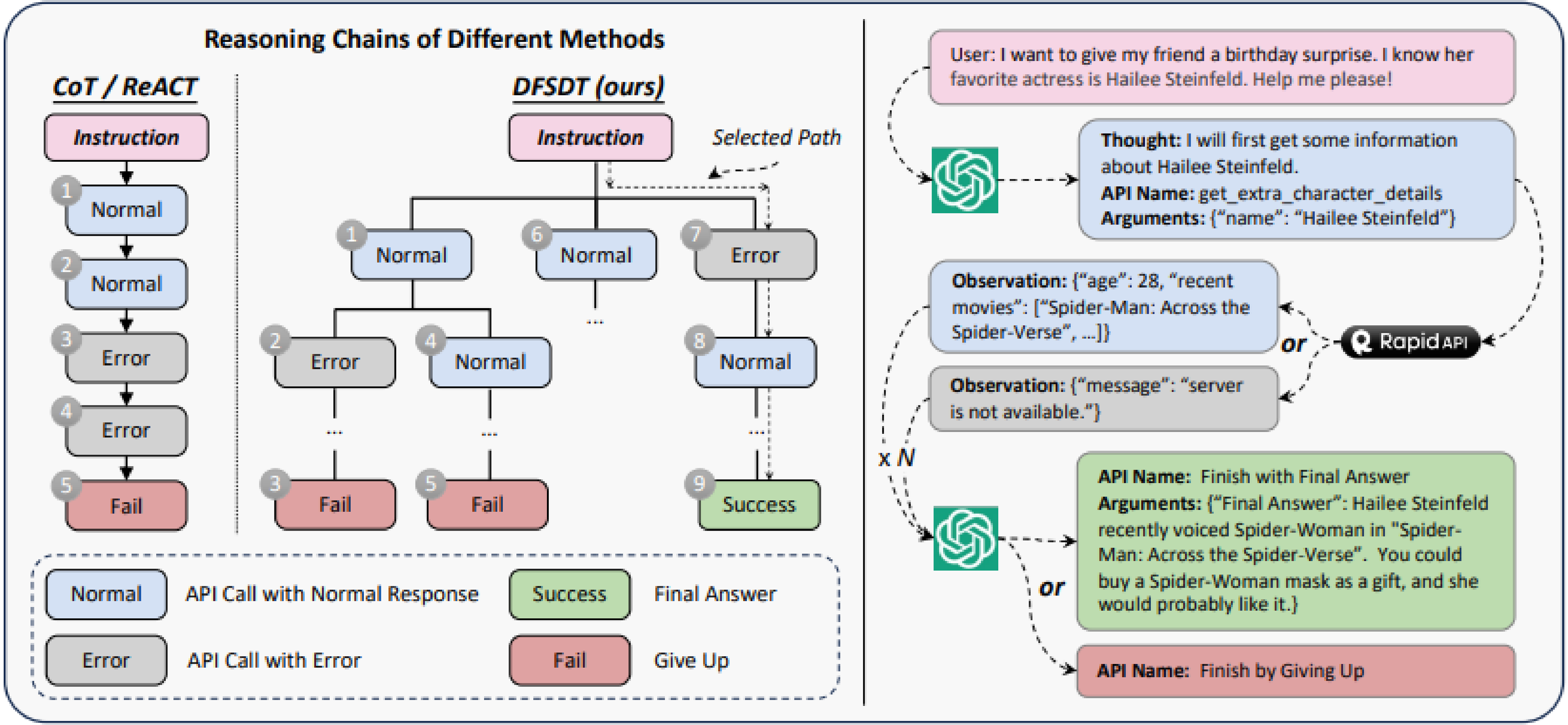
Quality Control & Scale

- Filter hallucinated links: drop instructions whose "relevant APIs" are not in S(sub N)
- Final dataset: ~200k (instruction, relevant-API) pairs (I1: 87,413 I2: 84,815 I3: 25,251)

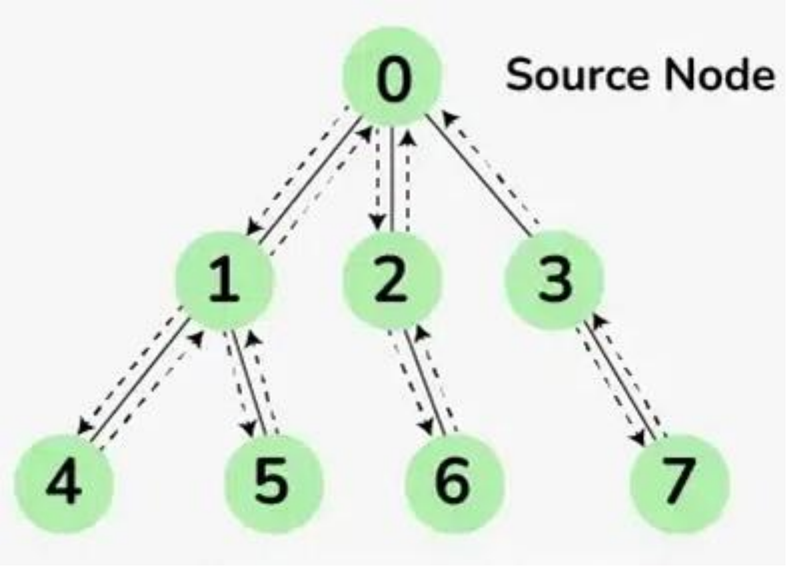
Diversity Evidence

- Human evaluation: high coverage & practicality
- Atlas visualization: supports diversity via clustering/coverage patterns

Solution Path Annotation



Depth First Search-based Decision Tree



Observed Issues:

- Error Propagation: a wrong step loops (mis-calling APIs, hallucinations).
- Limited Exploration: single trajectory → poor coverage of action space.

Depth-First Search Decision Tree:

- Allow model to evaluate different reasoning path.
    Proceeding along a promising path
    abandon existing node and expand a new one
- Prefer DFS over BFS: annotation finishes once one valid path is found; DFS is more cost-efficient than BFS

Result 126,486 pairs

Experiential: ToolEval

Why: APIs on RapidAPI change over time; an instruction can have many valid paths → no fixed ground truth; need consistent API versions; human eval is costly.

What: ChatGPT-based evaluator (AlpacaEval-style) with two metrics:

- Pass Rate – % of instructions successfully completed within a call/step budget (executability baseline).
- Win Rate – Given 1 instruction + 2 solution paths, ChatGPT prefers the better one using predefined criteria.

How: Use prompted criteria, run multiple trials, report averages to improve reliability.

Validity: High human alignment — 87.1% (Pass), 80.3% (Win) → scalable, fast, and model-agnostic evaluation without a single canonical solution path.

Efficacy of the API Retriever

Method	I1 NDCG		I2 NDCG		I3 NDCG		Average NDCG	
	@1	@5	@1	@5	@1	@5	@1	@5
BM25	18.4	19.7	12.0	11.0	25.2	20.4	18.5	17.0
Ada	57.5	58.8	36.8	30.7	54.6	46.8	49.6	45.4
Ours	**84.2**	**89.7**	**68.2**	**77.9**	**81.7**	**87.1**	**78.0**	**84.9**

Table 2: Our API retriever v.s. two baselines for three types of instructions (I1, I2, I3). We report NDCG@1 and NDCG@5.

Goal: Given an instruction, retrieve the most relevant APIs for downstream planning.

Method: Sentence-BERT bi-encoder dense retriever

- Encode instruction and API document into embeddings; score by embedding similarity.
- Training: positives = relevant APIs; negatives = sampled other APIs → contrastive learning.

Baselines: BM25, OpenAI text-embedding-ada-002.

Metric: NDCG@1 / NDCG@5 on I1 (single-tool), I2 (intra-category multi-tool), I3 (intra-collection multi-tool).

Result: Table

Conclusion: dense retrieval is feasible and effective. providing high-quality candidates

Superiority of DFSDT over ReACT

Metric: Pass Rate (ChatGPT judge)

ReACT@N: run ReACT repeatedly until total cost ≈ DFSDT; count pass once a valid path is found.

Method	I1	I2	I3	Average
ReACT	37.8	40.6	27.6	35.3
ReACT@N	49.4	49.4	34.6	44.5
DFSDT	**58.0**	**70.6**	**62.8**	**63.8**

Table 3: Pass rate of different reasoning strategies for three types of instructions (I1, I2, I3) based on ChatGPT.

- Under the same budget, DFSDT annotates more valid trajectories → lower total cost per accepted sample.
- Gains are larger on harder instructions (I2/I3) → expanding the search space solves cases where vanilla ReACT fails.
- Including these hard examples better elicits LLM tool-use capabilities for complex, real-world tasks.

Main Experiment

Model/Context:

Fine-tune LLaMA-2 7B; extend context from 4096 → 8192 via positional interpolation.

Generalization Levels:

Inst. (unseen instructions), Tool (unseen tools, seen category), Cat. (unseen categories).

Scenarios: I1 (single-tool), I2 (intra-category multi-tool), I3 (intra-collection multi-tool).

Setup:

- Default: Feed oracle APIs $S(Nsub)$ to all models;
- Reasoning: compare ReACT vs. DFSDT.
- Win Rate vs. ChatGPT-ReACT.test retriever setting:
- feed Top-5 retrieved APIs instead of oracle

Baselines: Vicuna, Alpaca, ChatGPT, Text-Davinci-003, GPT-4, Claude-2.

Metrics (ToolEval)

Main Result

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

- Vicuna/Alpaca = 0 (pass & win) → general dialog tuning ≠ tool-use competence.
- DFSDT > ReACT across models; Chat GPT+DFSDT ≥ GPT-4+ReACT (pass), comparable win.
- ToolLLaMA+DFSDT > Text-Davinci-003 / Claude-2; near ChatGPT, pass 2nd to GPT-4+DFSDT.
- With Top-5 retrieved APIs (vs. oracle set), ToolLLaMA improves further → retriever expands solution space and finds better substitutes.

Out-of-Distribution Generalization to APIBENCH

Set up

Domains: TorchHub, TensorHub, HuggingFace

Retrievers for ToolLLaMA: Our Retriever (dense) & Oracle Retriever

Baselines: Gorilla (LLaMA-7B) under ZS (zero-shot) and RS (retrieval-aware) settings

Metrics: AST accuracy (↑) & Hallucination rate (↓)

Out-of-Distribution Generalization to APIBENCH

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**

- ToolLLaMA + Our Retriever → higher AST than Gorilla + BM25 (both ZS/RS) on HuggingFace & TorchHub
- With Oracle Retriever, ToolLLaMA consistently > Gorilla-ZS across domains
- Dense retriever can reduce hallucinations and improve selection from a 16k+ API pool
- Gorilla does not transfer to ToolBench (multi-tool, multi-step), highlighting ToolLLaMA's planning streng

Related Work

Tool Learning:

- LLMs gain real-time knowledge, multimodality, and domain skills via tools;
- Open-source LLMs lag behind SOTA tool use; mechanisms remain unclear → ToolLLM bridges the gap.

Instruction Tuning vs. Tool Use:

- Self-instruct data boosts dialogue, but tool use is harder (vast APIs, multi-tool chains);
- Even GPT-4 often fails to find valid paths; prior tool datasets/pipelines don't meet real needs → ToolBench targets practical scenarios and improves data construction.

Prompting for Decision Making:

- ReAct integrates reasoning+acting but lacks retraction, causing error cascades;
- Reflexion adds self-correction; DFSDT generalizes further via branching search & backtracking;
- Related to Tree-of-Thought, but DFSDT targets open-ended decision spaces, not brute-forceable toy tasks.

Conclusion

ToolBench: 16k+ real REST APIs; diverse single- & multi-tool scenarios; ChatGPT-driven construction with minimal human effort.

DFSDT: Depth-first decision-tree reasoning (branching + backtracking) → stronger planning, executable paths for complex tasks.

ToolEval: Automatic Pass / Win evaluation with strong human alignment.

ToolLLaMA: LLaMA fine-tuned on ToolBench → near-ChatGPT performance; robust generalization to unseen APIs.

Neural API Retriever: Recommends relevant APIs; integrates with ToolLLaMA for a more automated tool-use pipeline.

OOD Generalization: Pipeline transfers to external domains (APIBench).

Reference List

Showed in Article

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

Bhargavi Paranjape et al.(2023)

Presentation by Mingrui Ye

Motivation and Problem

LLMs demonstrate emergent reasoning abilities in few- and zero-shot setups.

However, they struggle with multi-step reasoning and tool use, such as arithmetic, factual lookup, and programmatic reasoning.

Prior methods like Chain-of-Thought (CoT) or Toolformer:

- Rely on handcrafted prompts or fine-tuned models.
- Difficult to generalize to new tasks or tools.

Key Question:

How can we make LLMs automatically decompose complex problems and decide when to use tools, without retraining?

Compare with other methods

Chain-of-Thought (CoT) Prompting

- CoT and its variants (Least-to-Most, Self-Ask, AutoCoT) encourage LLMs to reason step by step.
- AutoCoT automatically generates reasoning chains, but remains free-form and lacks structured tool use.

Tool-Use Models

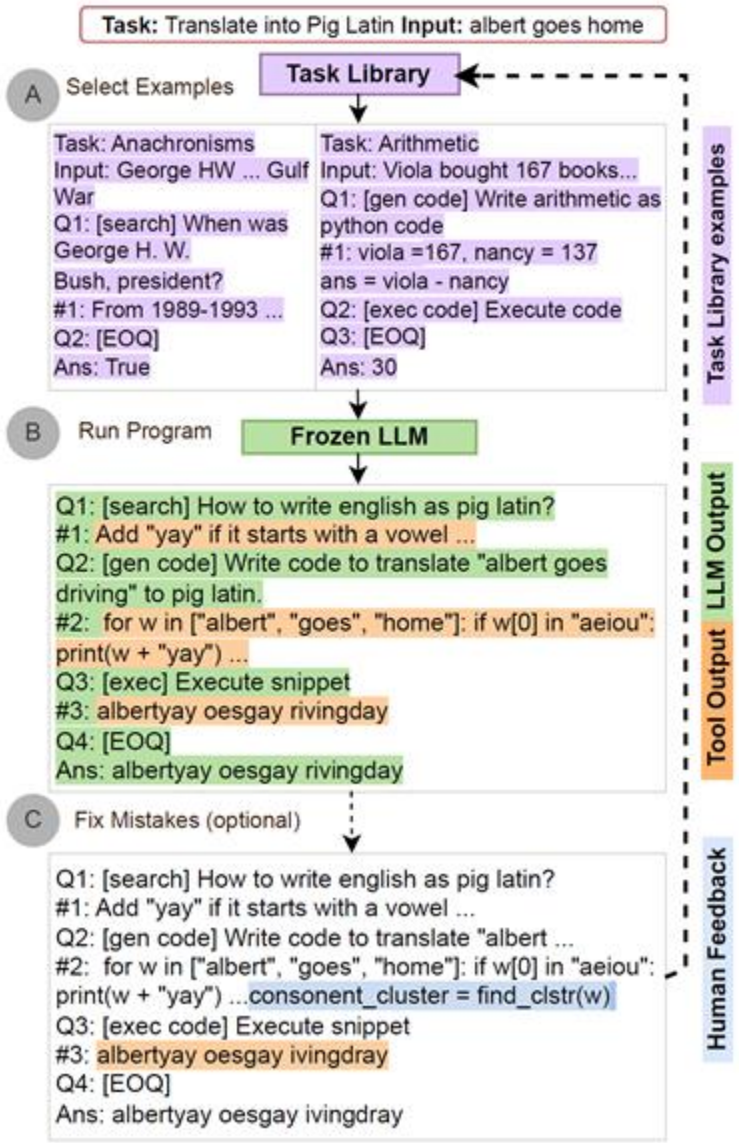
- Toolformer and similar methods fine-tune LLMs to call tools (search, calculator, translator).
- Require task-specific training and cannot easily extend to new tasks or tools.

Feature	CoT	Auto CoT	Tool-former	ART
Multi-step reasoning	✓	✓		✓
Limited supervision		✓	✓	✓
Tool use			✓	✓
Extendable libraries				✓
Cross-task transfer		✓	✓	✓
Human feedback	✓			✓

How ART Differs

- Automatic multi-step program generation without finetuning.
- Task & Tool libraries enable cross-task transfer and plug-and-play tools.
- Human feedback loop for error correction and continuous extension.

ART Architecture Overview



Task Library:

- Contains multi-step reasoning examples from 15 BigBench tasks.
- Each task = Input → multiple sub-steps (Q1/#1 …) → Final Answer.

Frozen LLM:

- Generates structured “programs” that integrate both text reasoning and symbolic computation.

Tool Library:

- Tools: Search, Code Generation, Code Execution, Lookup, Prolog Engine.
- Each tool corresponds to a symbolic tag [tool_name].

Human Feedback (Optional):

- Users can edit reasoning chains, add new tools, or correct errors.

How ART Works

Step-by-Step Process:

1. **Task Retrieval:** ART retrieves similar tasks from the library based on textual similarity or small validation set.
2. **Program Generation:** LLM writes structured multi-step reasoning using Qn: [tool] ... #n: ... format.
3. **Tool Execution:** ART pauses at tool symbols (e.g., [search], [exec code]), executes, and inserts output.
4. **Result Integration:** LLM continues reasoning using results from tool calls.
5. **Optional Feedback Loop:** Users can modify a reasoning chain to correct logic or add missing steps.

Task Library Design

1. The Task Library build from 15 representative BigBench tasks, covering five reasoning clusters:

Cluster	Representative	Capability
Arithmetic	GSM8K, Aqua-Rat	arithmetic and algebra problems
Code	Auto Debugging	Generating and executing python code
Search and question decomposition	Anachronisms, Musique	Single or multi-step questions that require search
Free-form reasoning	Hyperbation, Formal Fallacies	Explaining step-by-step reasoning in natural language
String Operations	Language games, Date Understanding	Reformatting/editing strings, checking string entailment, etc

2. Uses a Parsing Expression Grammar (PEG) to define program structure.

3. Enables easy parsing, tool calling, and resumption of generation.

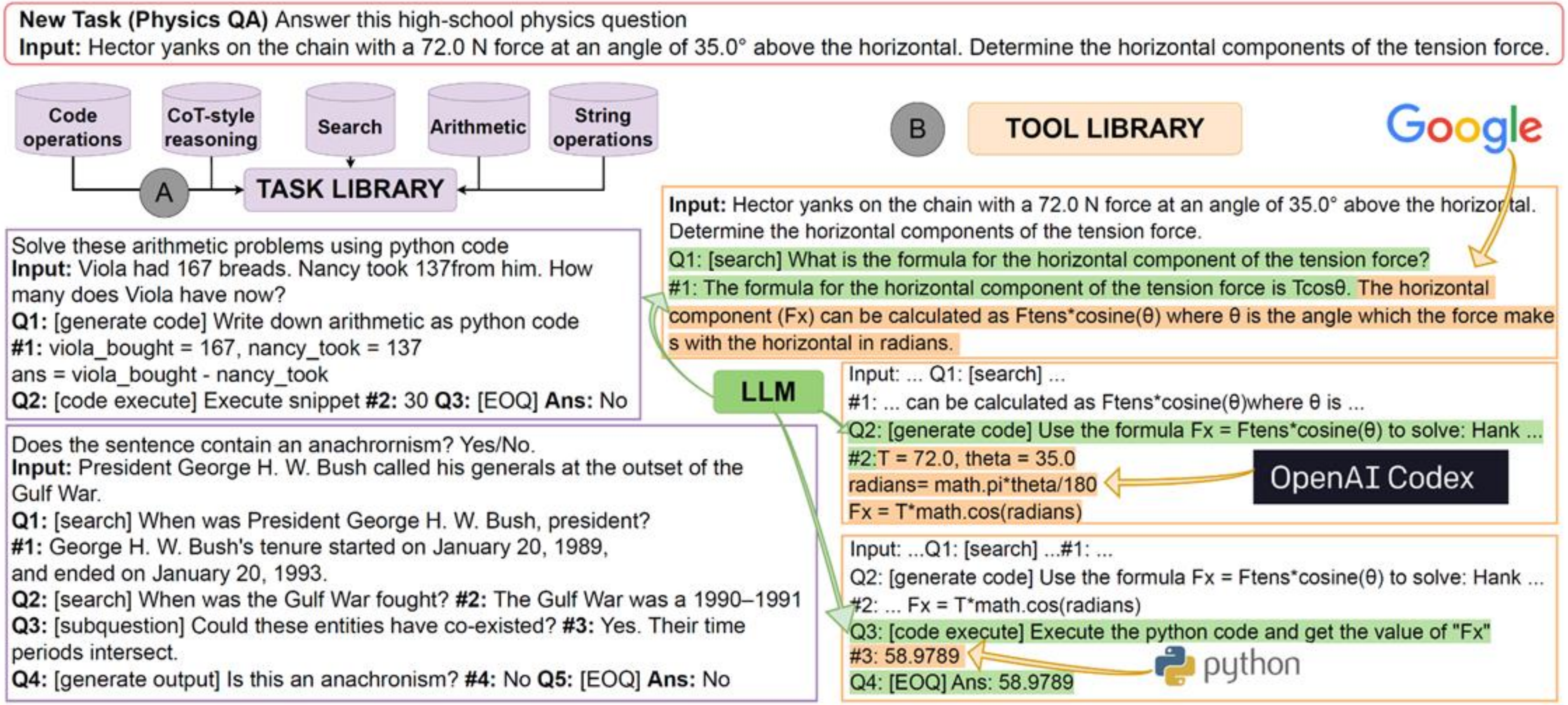
Tool Library

Core Tools:

- Search – via SerpAPI (Google API)
- Code Generation – via Codex
- Code Execution – runs Python snippets in sandbox
- Lookup – dictionary/knowledge base queries
- Prolog Engine – logic reasoning in formal fallacy tasks

Key Benefit:
Easily extendable — add new tools or modify existing ones without retraining the LLM.

Example: Physics QA



Human Feedback

ART is designed to naturally accept human feedback without any finetuning. Because reasoning is expressed as interpretable multi-step programs, users can directly edit or debug them.

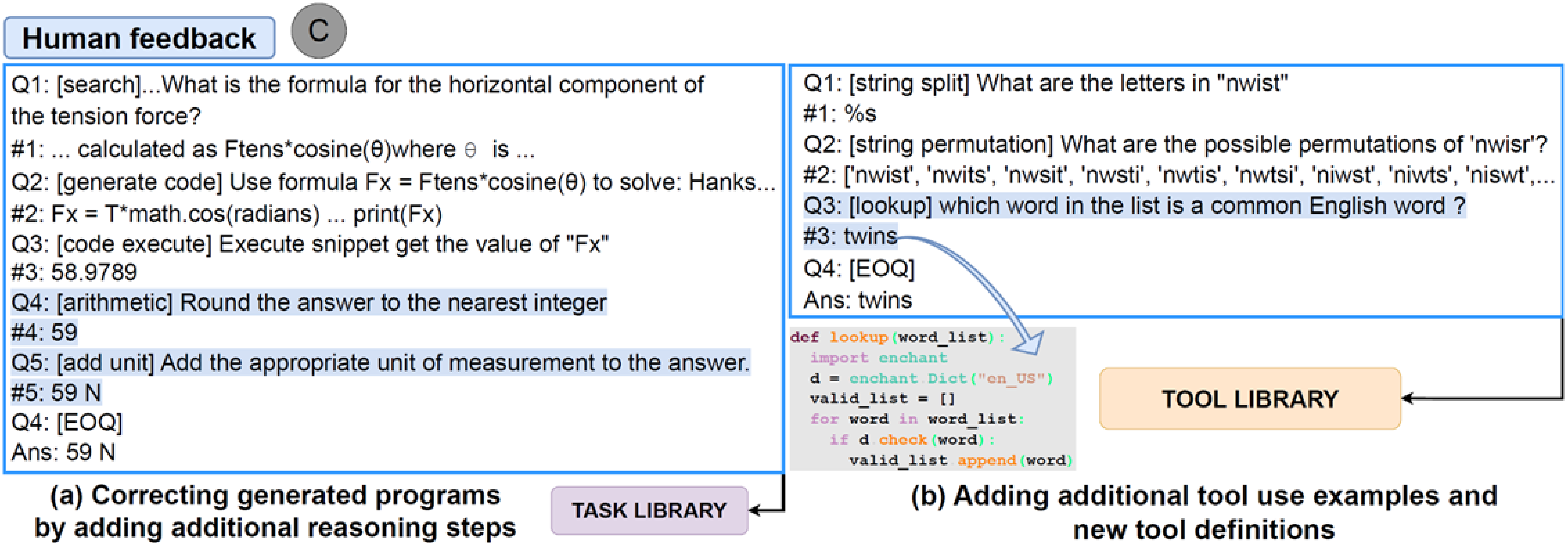
Forms of Feedback

- Editing task or tool libraries — users can instantly modify stored examples or tool APIs.
- Program debugging — instead of rewriting from scratch, users fix parts of an existing

Users can modify the reasoning chain:

- Correct incorrect sub-step outputs.
- Add / remove sub-steps with proper inputs and answers.
- Introduce calls to new tools (e.g., [lookup], [add unit]).

Human Feedback Examples



Experimental Setup

Datasets:

- 15 BigBench training tasks (for library)
- 19 unseen BigBench test tasks
- 6 MMLU tasks (for cross-benchmark validation)
- Toolformer-style QA tasks (SQuAD, TriviaQA, SVAMP, MAWPS)

Baselines:

- Few-shot prompting
- AutoCoT (automatic chain-of-thought)
- ART without tool use
- Best GPT-3(175B)/Toolformer results

Models:

- LLM: InstructGPT (text-davinci-002)
- Code generator: Codex
- Temperature = 0.3

Results on Task Library

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

- ART performs on par or better than Auto-CoT and Few-Shot baselines across all clusters.
- Especially strong on Arithmetic.
- From ART and ART w/o Tool Use. Shows that ART successfully learns structured, interpretable reasoning sequences within known task types.

Results on Test Tasks

Task Name (Cluster)	Few Shot	AutoCot	ART w/o Tool Use	ART	GPT-3 Best
Test Tasks					
Sentence Ambiguity (Search)	70.67[5]	51.47	71.00	73.33	-
Strategy QA (Search)	55.49[5]	27.22	59.37	66.44	-
Physics (Search)	70.09[5]	61.83	59.13	67.55	-
Δ with ART (Search)	+3.7	+22.27	+ 5.9		
Physics Questions (Arithmetic)	7.02[5]	5.56	6.30	20.37	-
Operators (Arithmetic)	71.23[7]	75.52	71.80	92.00	-
Unit interpretation (Arithmetic)	58.2[7]	41.20	51.4	53.99	-
Repeat copy logic (Arithmetic)	50.01[7]	15.63	31.25	44.38	-
Object Counting (Arithmetic)	39.2[7]	26.80	42.2	87.00	81.20[1]
Penguins in a table (Arithmetic)	58.23[7]	40.40	68.86	77.85	72.34[1]
Reasoning about objects (Arithmetic)	71.00[7]	33.33	45.35	64.34	52.69[1]
Tracking shuffled objects (Arithmetic)	22.39[7]	19.44	18.14	37.67	36.32[1]
Δ with ART (Arithmetic)	+19.0	+36.7	+ 23.1		+6.1
Word Unscramble (String)	40.72[7]	32.44	23.03	42.7	-
Simple Text Editing (Code)	35.31[5]	30.21	20.74	27.65	-
CS Algorithms (Code)	73.48[7]	0.0	41.59	88.11	-
Sports Understanding (CoT)	69.74[5]	51.47	92.89	-	86.59[1]
Snarks (CoT)	54.58[5]	57.24	57.13	-	65.2[1]
Disambiguation QA (Free-form)	55.03[5]	48.45	55.89	-	60.62[1]
Temporal sequences (CoT)	55.80[7]	19.70	49.5	-	81.8[1]
Ruin names (CoT)	71.01[5]	55.28	60.22	-	-
Δ with ART (Misc)	2.4	22.5	24.37		-9.4
Δ with ART (Overall)	+6.9	+24.6	+16.7		-1.7
MMLU					
College Computer Science (Search)	41.00	43.99	63.40	67.80	63.6[6]
Astronomy (Search)	62.10	41.48	76.71	79.1	62.5[6]
Business Ethics (Search)	61.60	48.8	77.17	81.16	72.7[6]
Virology (Search)	50.03	49.52	71.60	71.49	50.72[6]
Geography (Search)	77.67	57.07	70.30	71.71	81.8[6]
Mathematics (Arithmetic)	36.67	33.77	39.50	45.66	34.5[6]
Δ with ART (MMLU)	+14.6	+23.7	+3.0		+8.5

BigBench test tasks:

- ART outperforms few-shot learning (6.9 % points). In particular, ART has significant improvements on arithmetic tasks (+19.0) and is comparable to the few-shot performance on search tasks.
- ART is better than AutoCoT on almost all tasks (24.6% points).
- Compare with GPT-3 Best, ART performs favorably on average, especially on arithmetic tasks (+6.1 % points).

Other benchmarks(MMLU):

- ART is more effective than all baselines on 5/6 tasks

Improve ART with Self-Consistency

	Simple Text Editing	CS Algorithms	Strategy QA	Physics Questions	Unit Interpretation	Reasoning about colored objects
ART	27.65	88.11	66.44	20.37	53.99	64.34
+ Self Consistency	30.67(+3.0)	90.99(+2.9)	70.76(+4.3)	24.07(+3.7)	57.20(+3.2)	69.11(+4.8)

In this table we can see: Self-consistency smooths stochastic reasoning errors, yielding +3 ~ 5 percentage points improvement with no retraining.

Improve ART with Human Feedback

Task	CoT	+Human	ART	+ Human	GPT-3 Best	Human Feedback
CS Algorithms	0.0	23.0	88.11	92.73	73.48	C: longest common subsequence code
Reasong about objs.	33.33	67.75	64.34	98.90	71.00	C: Define object, color, count data structure
Repeat Copy Logic*	15.63	45.22	44.38	80.31	50.01	C: string edit operation
Sentence Ambiguity	51.47	72.33	73.33	83.67	70.67	C: Constrain queries to extract relevant info.
Simple Text editing*	30.21	35.31	27.65	36.11	35.31	C: string edit operation
Strategy QA*	27.22	29.19	66.44	69.15	55.49	C: Constrain queries to extract relevant info.
Physics*	61.83	68.21	67.55	72.55	70.09	A: [search] Formula that connects mass, ...
Temporal Sequences	19.70	30.22	49.5	88.00	81.8	A: [subquestion] Is X free Yam to Zam?
Track Shuffled objs.	19.44	36.48	37.67	99.86	36.32	C: Define object pair data struct, swap logic
Unit Interpretation*	41.2	41.2	53.99	95.0	58.2	A: [add unit] Add the right unit to the answer
Word Unscrambling*	32.44	33.40	42.70	62.11	40.72	T: lookup permutations in dictionary
Average	30.2	**43.8**	56.0	**79.85**	58.5	

In most of the task, human feedback can drastically boost performance — up to +38 points on some tasks without any fine-tuning or model retraining.

Limitation and Future Work

Limitations

- Task Library Dependence: Performance tied to the quality of stored examples.
- Error Propagation: Early-step mistakes cascade through reasoning chains.
- Limited Tool Diversity: Current tools (search, code, lookup) restrict scope.
- Execution Instability: External calls may fail; needs safer sandboxing.
- Narrow Evaluation: Tested mainly on BigBench and MMLU.

Future Work

- Expand Tools & Tasks: Add vision, retrieval, and simulation APIs.
- Self-Correction: Integrate reflection or verifier modules.
- Human Feedback Loop: Support real-time editing and improvement.
- Broader Testing: Validate on open-domain and multimodal reasoning.

Conclusion

Summary of Findings：

- ART reframes reasoning as program synthesis — combining natural language and tool use.
- It learns structured, interpretable multi-step programs within its task library.
- It generalizes these reasoning programs to unseen BigBench and MMLU tasks without any fine-tuning.
- External tools (search, code execution, lookup) further amplify reasoning accuracy.

Overall Insight：

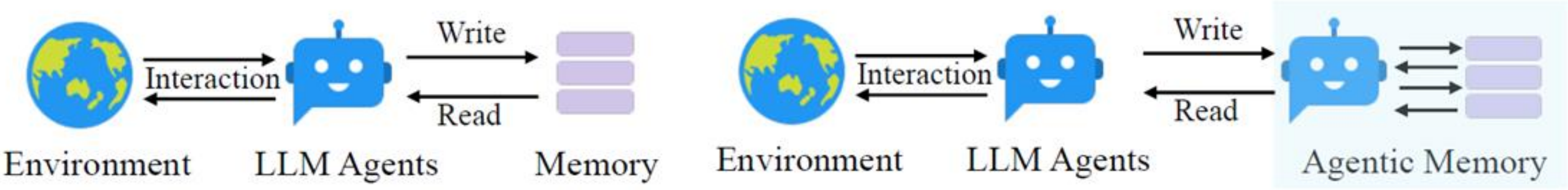
ART demonstrates that “Prompt = Program = Reasoning”：large language models can plan, execute, and improve reasoning pipelines automatically, paving the way toward autonomous tool-using AI systems.

A-Mem: Agentic Memory for LLM Agents

Wujiang Xu et al., Meta AI (2023)

Presentation by Mingrui Ye, Deyuan Yang, Yancheng Jin

Introduction



(a) Traditional memory system. (b) Our proposed agentic memory.

- Traditional LLM memory systems rely on fixed read/write rules, making them rigid and hard to adapt to new tasks.
- A-MEM (Agentic Memory) introduces dynamic, self-organizing memory. It allows agents to autonomously store, link, and evolve information instead of following preset workflows.

- A-MEM based on the Zettelkasten method — each interaction becomes a structured “note” (content, keywords, tags, embedding). Enable long-term reasoning and continuous learning through adaptive memory evolution.

Related Work

Memory for LLM Agents

- Early works (MemGPT, MemoryBank, ReadAgent, SCM) → provide storage but rely on predefined read/write rules.
- Limitations: rigid workflows, poor adaptability across new environments.

Retrieval-Augmented Generation (RAG)

- Enhances LLMs via retrieving external knowledge before generation.
- “Agentic RAG” adds autonomy in when and what to retrieve (e.g., Self-RAG, Active-RAG).
- A-MEM vs RAG:
  - **RAG = agency during retrieval only.**
  - **A-MEM = agency in storage + evolution, forming a self-organizing memory graph.**

Method: How A-Mem Works

1. Note Construction: Turn interactions into rich, structured notes
2. Link Generation: Automatically find connections betweens notes
3. Memory Evolution: Update old memories with new insights
4. Memory Retrieval: retrieve relevant historical context for better understanding

This creates a living, interconnected knowledge network.

Step 1: Note Construction - Creating Rich Memories

$m_i = \{c_i, t_i, K_i, G_i, X_i, e_i, L_i\}$ memory note (for every interaction)

$K_i, G_i, X_i \leftarrow \mathrm{LLM}(c_i \,\|\, t_i \,\|\, P_{s1})$ Use LLM generate Ki, Gi, Xi for deeper understanding beyond raw text

$e_i = f_{\mathrm{enc}}[\ \mathrm{concat}(c_i, K_i, G_i, X_i)\ ]$ text encoder that encapsulates all textual components of note

c_i -raw content

t_i -timestamp

K_i -LLM-generated Keywords

G_i -LLM-generated Tags

X_i -LLM-generated Contextual Description

e_i -Dense Vector Embedding (for similarity search)

L_i -Links to other memories

Step 2: Link Generation

$$s_{n,j} = \frac{e_n \cdot e_j}{|e_n||e_j|}$$

Similarity score

$$\mathcal{M}_{\text{near}}^{n} = \{m_j \,|\, \text{rank}(s_{n,j}) \leq k, m_j \in \mathcal{M}\}$$

$$L_i \leftarrow \text{LLM}(m_n \,||\mathcal{M}_{\text{near}}^{n} \,||P_{s2})$$

- Use the embedding e_i to find top k similar historical memories
- Use an LLM to decide which of these should be formally linked based on shared context and attributes

Step 3: Memory Evolution

$$m_j^* \leftarrow \text{LLM}(m_n \, \| \mathcal{M}_{\text{near}}^n \setminus m_j \, \| m_j \, \| P_{s3})$$

evolution process

- For each of the top-k similar memories, the LLM analyzes the new memory
- It can update the context, keywords and tags of the old memory to reflect new understanding

Step 2 and Step 3 is the “Agent” part: The memory system actively reasons about the restructures itself.

Step 4: Retrieve Relative Memory

$$e_q = f_{\text{enc}}(q)$$ dense vector for same encoder

$$s_{q,i} = \frac{e_q \cdot e_i}{|e_q||e_i|}, \text{where } e_i \in m_i, \ \forall m_i \in \mathcal{M}$$ Similarity score

$$\mathcal{M}_{\text{retrieved}} = \{m_i | \text{rank}(s_{q,i}) \le k, m_i \in \mathcal{M}\}$$ Top k memory retrieved

- provide relevant historical context that improves agent understanding and response
- Connect current interaction with past experience

Experiment Setup

Datasets:

- LoCoMo (long conversations)
  - **Purpose: evaluate long-term conversational memory**
  - **Key feature: very long conversations (avg. 9K tokens, up to 35 sessions)**
  - **Question Types: Test different reasoning skills**
    - Single-Hop (one session)
    - Multi-Hop (across sessions)
    - Temporal Reasoning
    - Open-Domain
    - Adversarial
- DialSim (TV show dialogues)
  - **Purpose: Evaluate understanding of long-term, multi-party dialogues**
  - **Source: Derived from TV shows**
  - **Scale: ~350,000 tokens, over 1,000 questions**

Experiment Setup

Implementation Details:

- Fair Comparison
  - **All methods used identical system prompts to ensure fairness**

- Model Deployment
  - **Local Models (Qwen, Llama): run locally using Ollama.**
  - **Structured Outputs: Managed by LiteLLM framework**
  - **GPT Models: used the official OpenAI API**

- Key Parameters
  - **Retrieval (k-value): Primarily used k=10 for efficiency, with adjustments for specific tasks**
  - **Embedding Model: Used all-minilm-16-v2 for all text embeddings across all experiments.**

Experimental Setup

Baseline and Matrics

- Baselines:
  - LocoMo: uses the full conversation history as context (very expensive)
  - MemGPT: A sophisticated memory system with a context hierarchy
  - MemoryBank: Manages memory using a forgetting curve theory
  - ReadAgent: uses a pagination and “gisting” strategy for long documents

- Evaluation Metrics:
  - F1 score: measures answer accuracy (balance of precision and recall)
  - BLEU-1: Measures word-overlap with the correct answer

Performance Analysis

Across Models:

- Non-GPT models (Qwen, LLaMA): A-MEM outperforms all baselines in every category.
- GPT models: LoCoMo / MemGPT strong in simple fact retrieval, but A-MEM 2× better on Multi-Hop reasoning.

Cross-Dataset Validation:

- On DialSim, A-MEM achieves F1 = 3.45 (+35% vs LoCoMo, +192% vs MemGPT).

Table 1: Experimental results on LoCoMo dataset of QA tasks across five categories (Multi Hop, Temporal, Open Domain, Single Hop, and Adversial) using different methods. Results are reported in F1 and BLEU-1 (%) scores. The best performance is marked in bold, and our proposed method A-MEM (highlighted in gray) demonstrates competitive performance across six foundation language models.

Model		Method	Category										Average		
			Multi Hop		Temporal		Open Domain		Single Hop		Adversial		Ranking		Token
			F1	BLEU	F1	BLEU	F1	BLEU	F1	BLEU	F1	BLEU	F1	BLEU	Length
GPT	4o-mini	LoCoMo	25.02	19.75	18.41	14.77	12.04	11.16	40.36	29.05	**69.23**	**68.75**	2.4	2.4	16,910
		ReadAgent	9.15	6.48	12.60	8.87	5.31	5.12	9.67	7.66	9.81	9.02	4.2	4.2	643
		MemoryBank	5.00	4.77	9.68	6.99	5.56	5.94	6.61	5.16	7.36	6.48	4.8	4.8	432
		MemGPT	26.65	17.72	25.52	19.44	9.15	7.44	41.04	34.34	43.29	42.73	2.4	2.4	16,977
		A-Mem	**27.02**	**20.09**	**45.85**	**36.67**	**12.14**	**12.00**	**44.65**	**37.06**	50.03	49.47	**1.2**	**1.2**	2,520
	4o	LoCoMo	28.00	18.47	9.09	5.78	16.47	14.80	**61.56**	**54.19**	**52.61**	**51.13**	2.0	2.0	16,910
		ReadAgent	14.61	9.95	4.16	3.19	8.84	8.37	12.46	10.29	6.81	6.13	4.0	4.0	805
		MemoryBank	6.49	4.69	2.47	2.43	6.43	5.30	8.28	7.10	4.42	3.67	5.0	5.0	569
		MemGPT	30.36	22.83	17.29	13.18	12.24	11.87	60.16	53.35	34.96	34.25	2.4	2.4	16,987
		A-Mem	**32.86**	**23.76**	**39.41**	**31.23**	**17.10**	**15.84**	48.43	42.97	36.35	35.53	**1.6**	**1.6**	1,216
Qwen2.5	1.5b	LoCoMo	9.05	6.55	4.25	4.04	9.91	8.50	11.15	8.67	40.38	40.23	3.4	3.4	16,910
		ReadAgent	6.61	4.93	2.55	2.51	5.31	12.24	10.13	7.54	5.42	27.32	4.6	4.6	752
		MemoryBank	11.14	8.25	4.46	2.87	8.05	6.21	13.42	11.01	36.76	34.00	2.6	2.6	284
		MemGPT	10.44	7.61	4.21	3.89	13.42	11.64	9.56	7.34	31.51	28.90	3.4	3.4	16,953
		A-Mem	**18.23**	**11.94**	**24.32**	**19.74**	**16.48**	**14.31**	**23.63**	**19.23**	**46.00**	**43.26**	**1.0**	**1.0**	1,300
	3b	LoCoMo	4.61	4.29	3.11	2.71	4.55	5.97	7.03	5.69	16.95	14.81	3.2	3.2	16,910
		ReadAgent	2.47	1.78	3.01	3.01	5.57	5.22	3.25	2.51	15.78	14.01	4.2	4.2	776
		MemoryBank	3.60	3.39	1.72	1.97	6.63	6.58	4.11	3.32	13.07	10.30	4.2	4.2	298
		MemGPT	5.07	4.31	2.94	2.95	7.04	7.10	7.26	5.52	14.47	12.39	2.4	2.4	16,961
		A-Mem	**12.57**	**9.01**	**27.59**	**25.07**	**7.12**	**7.28**	**17.23**	**13.12**	**27.91**	**25.15**	**1.0**	**1.0**	1,137
Llama 3.2	1b	LoCoMo	11.25	9.18	7.38	6.82	11.90	10.38	12.86	10.50	51.89	48.27	3.4	3.4	16,910
		ReadAgent	5.96	5.12	1.93	2.30	12.46	11.17	7.75	6.03	44.64	40.15	4.6	4.6	665
		MemoryBank	13.18	10.03	7.61	6.27	15.78	12.94	17.30	14.03	52.61	47.53	2.0	2.0	274
		MemGPT	9.19	6.96	4.02	4.79	11.14	8.24	10.16	7.68	49.75	45.11	4.0	4.0	16,950
		A-Mem	**19.06**	**11.71**	**17.80**	**10.28**	**17.55**	**14.67**	**28.51**	**24.13**	**58.81**	**54.28**	**1.0**	**1.0**	1,376
	3b	LoCoMo	6.88	5.77	4.37	4.40	10.65	9.29	8.37	6.93	30.25	28.46	2.8	2.8	16,910
		ReadAgent	2.47	1.78	3.01	3.01	5.57	5.22	3.25	2.51	15.78	14.01	4.2	4.2	461
		MemoryBank	6.19	4.47	3.49	3.13	4.07	4.57	7.61	6.03	18.65	17.05	3.2	3.2	263
		MemGPT	5.32	3.99	2.68	2.72	5.64	5.54	4.32	3.51	21.45	19.37	3.8	3.8	16,956
		A-Mem	**17.44**	**11.74**	**26.38**	**19.50**	**12.53**	**11.83**	**28.14**	**23.87**	**42.04**	**40.60**	**1.0**	**1.0**	1,126

Table 2: Comparison of different memory mechanisms across multiple evaluation metrics on DialSim [16]. Higher scores indicate better performance, with A-MEM showing superior results across all metrics.

Method	F1	BLEU-1	ROUGE-L	ROUGE-2	METEOR	SBERT Similarity
LoCoMo	2.55	3.13	2.75	0.90	1.64	15.76
MemGPT	1.18	1.07	0.96	0.42	0.95	8.54
A-Mem	**3.45**	**3.37**	**3.54**	**3.60**	**2.05**	**19.51**

Cost-Efficiency Analysis

Token Usage: ~1.2 K tokens per operation (-85–93% vs baselines ~16.9 K).

API Cost: <$0.0003 per operation → economical large-scale deployment.

Runtime: 5.4 s (GPT-4o-mini), 1.1 s (local LLaMA 3.2 1B).

Efficiency Balance: Despite multiple LLM calls, A-MEM keeps low cost while doubling multi-hop performance.

Takeaway: Efficient and scalable for real-world LLM agents.

Table 1: Experimental results on LoCoMo dataset of QA tasks across five categories (Multi Hop, Temporal, Open Domain, Single Hop, and Adversial) using different methods. Results are reported in F1 and BLEU-1 (%) scores. The best performance is marked in bold, and our proposed method A-MEM (highlighted in gray) demonstrates competitive performance across six foundation language models.

Model		Method	Multi Hop F1	Multi Hop BLEU	Temporal F1	Temporal BLEU	Open Domain F1	Open Domain BLEU	Single Hop F1	Single Hop BLEU	Adversial F1	Adversial BLEU	Ranking F1	Ranking BLEU	Token Length
GPT	4o-mini	LoCoMo	25.02	19.75	18.41	14.77	12.04	11.16	40.36	29.05	**69.23**	**68.75**	2.4	2.4	16,910
		ReadAgent	9.15	6.48	12.60	8.87	5.31	5.12	9.67	7.66	9.81	9.02	4.2	4.2	643
		MemoryBank	5.00	4.77	9.68	6.99	5.56	5.94	6.61	5.16	7.36	6.48	4.8	4.8	432
		MemGPT	26.65	17.72	25.52	19.44	9.15	7.44	41.04	34.34	43.29	42.73	2.4	2.4	16,977
		A-Mem	**27.02**	**20.09**	**45.85**	**36.67**	**12.14**	**12.00**	**44.65**	**37.06**	50.03	49.47	**1.2**	**1.2**	2,520
	4o	LoCoMo	28.00	18.47	9.09	5.78	16.47	14.80	**61.56**	**54.19**	**52.61**	**51.13**	2.0	2.0	16,910
		ReadAgent	14.61	9.95	4.16	3.19	8.84	8.37	12.46	10.29	6.81	6.13	4.0	4.0	805
		MemoryBank	6.49	4.69	2.47	2.43	6.43	5.30	8.28	7.10	4.42	3.67	5.0	5.0	569
		MemGPT	30.36	22.83	17.29	13.18	12.24	11.87	60.16	53.35	34.96	34.25	2.4	2.4	16,987
		A-Mem	**32.86**	**23.76**	**39.41**	**31.23**	**17.10**	**15.84**	48.43	42.97	36.35	35.53	**1.6**	**1.6**	1,216
Qwen2.5	1.5b	LoCoMo	9.05	6.55	4.25	4.04	9.91	8.50	11.15	8.67	40.38	40.23	3.4	3.4	16,910
		ReadAgent	6.61	4.93	2.55	2.51	5.31	12.24	10.13	7.54	5.42	27.32	4.6	4.6	752
		MemoryBank	11.14	8.25	4.46	2.87	8.05	6.21	13.42	11.01	36.76	34.00	2.6	2.6	284
		MemGPT	10.44	7.61	4.21	3.89	13.42	11.64	9.56	7.34	31.51	28.90	3.4	3.4	16,953
		A-Mem	**18.23**	**11.94**	**24.32**	**19.74**	**16.48**	**14.31**	**23.63**	**19.23**	**46.00**	**43.26**	**1.0**	**1.0**	1,300
	3b	LoCoMo	4.61	4.29	3.11	2.71	4.55	5.97	7.03	5.69	16.95	14.81	3.2	3.2	16,910
		ReadAgent	2.47	1.78	3.01	3.01	5.57	5.22	3.25	2.51	15.78	14.01	4.2	4.2	776
		MemoryBank	3.60	3.39	1.72	1.97	6.63	6.58	4.11	3.32	13.07	10.30	4.2	4.2	298
		MemGPT	5.07	4.31	2.94	2.95	7.04	7.10	7.26	5.52	14.47	12.39	2.4	2.4	16,961
		A-Mem	**12.57**	**9.01**	**27.59**	**25.07**	**7.12**	**7.28**	**17.23**	**13.12**	**27.91**	**25.15**	**1.0**	**1.0**	1,137
Llama 3.2	1b	LoCoMo	11.25	9.18	7.38	6.82	11.90	10.38	12.86	10.50	51.89	48.27	3.4	3.4	16,910
		ReadAgent	5.96	5.12	1.93	2.30	12.46	11.17	7.75	6.03	44.64	40.15	4.6	4.6	665
		MemoryBank	13.18	10.03	7.61	6.27	15.78	12.94	17.30	14.03	52.61	47.53	2.0	2.0	274
		MemGPT	9.19	6.96	4.02	4.79	11.14	8.24	10.16	7.68	49.75	45.11	4.0	4.0	16,950
		A-Mem	**19.06**	**11.71**	**17.80**	**10.28**	**17.55**	**14.67**	**28.51**	**24.13**	**58.81**	**54.28**	**1.0**	**1.0**	1,376
	3b	LoCoMo	6.88	5.77	4.37	4.40	10.65	9.29	8.37	6.93	30.25	28.46	2.8	2.8	16,910
		ReadAgent	2.47	1.78	3.01	3.01	5.57	5.22	3.25	2.51	15.78	14.01	4.2	4.2	461
		MemoryBank	6.19	4.47	3.49	3.13	4.07	4.57	7.61	6.03	18.65	17.05	3.2	3.2	263
		MemGPT	5.32	3.99	2.68	2.72	5.64	5.54	4.32	3.51	21.45	19.37	3.8	3.8	16,956
		A-Mem	**17.44**	**11.74**	**26.38**	**19.50**	**12.53**	**11.83**	**28.14**	**23.87**	**42.04**	**40.60**	**1.0**	**1.0**	1,126

Table 2: Comparison of different memory mechanisms across multiple evaluation metrics on DialSim [16]. Higher scores indicate better performance, with A-MEM showing superior results across all metrics.

Method	F1	BLEU-1	ROUGE-L	ROUGE-2	METEOR	SBERT Similarity
LoCoMo	2.55	3.13	2.75	0.90	1.64	15.76
MemGPT	1.18	1.07	0.96	0.42	0.95	8.54
A-Mem	**3.45**	**3.37**	**3.54**	**3.60**	**2.05**	**19.51**

Ablantion Study

Setup: Remove modules to test contribution

- No LG & ME → largest drop; memory lacks structure
- No ME (LG only) → intermediate performance
- Full A-MEM → best across all categories

Result (GPT-4o-mini base):

- Multi-Hop F1: 9.65 (w/o LG&ME) → 21.35 (w/o ME) → 27.02 (Full)
- Open-Domain F1: 7.77 → 10.13 → 12.14
- Temporal F1: 24.55 → 31.24 → 45.85
- Adversarial F1: 15.32 → 44.16 → 50.03

Takeaways:

- LG = foundation (builds the memory graph; big gains already)
- ME = refinement (evolves/updates notes; pushes to SOTA)
- LG + ME are complementary → effective, scalable memory system

Method	Multi Hop		Temporal		Open Domain		Single Hop		Adversial	
	F1	BLEU-1	F1	BLEU-1	F1	BLEU-1	F1	BLEU-1	F1	BLEU-1
w/o LG & ME	9.65	7.09	24.55	19.48	7.77	6.70	13.28	10.30	15.32	18.02
w/o ME	21.35	15.13	31.24	27.31	10.13	10.85	39.17	34.70	44.16	45.33
A-MEM	**27.02**	**20.09**	**45.85**	**36.67**	**12.14**	**12.00**	**44.65**	**37.06**	**50.03**	**49.47**

Hyperparameter Analysis

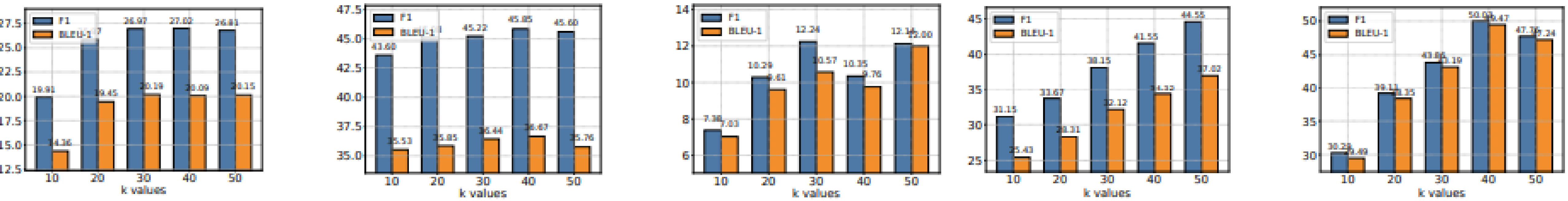
Goal: Examine impact of retrieval parameter k (10–50).

Setup: GPT-4o-mini base; 5 task types (Multi-Hop, Temporal, Open-Domain, Single-Hop, Adversarial).

Findings:

- Performance ↑ as k increases, then plateaus or drops.
- Most visible in Multi-Hop & Open-Domain tasks.
- Larger k = richer context but more noise / longer processing.

Conclusion: Moderate k (10–20) offers the best trade-off between context richness and efficiency.



(a) Multi Hop (b) Temporal (c) Open Domain (d) Single Hop (e) Adversarial

Scaling Analysis

Setup: Compare A-MEM with MemoryBank & ReadAgent using identical data.

Memory sizes: 1K → 10K → 100K → 1M entries (×10 each step).

Memory Size	Method	Memory Usage (MB)	Retrieval Time (μs)
1,000	A-MEM	1.46	0.31 ± 0.30
	MemoryBank [39]	1.46	0.24 ± 0.20
	ReadAgent [17]	1.46	43.62 ± 8.47
10,000	A-MEM	14.65	0.38 ± 0.25
	MemoryBank [39]	14.65	0.26 ± 0.13
	ReadAgent [17]	14.65	484.45 ± 93.86
100,000	A-MEM	146.48	1.40 ± 0.49
	MemoryBank [39]	146.48	0.78 ± 0.26
	ReadAgent [17]	146.48	6,682.22 ± 111.63
1,000,000	A-MEM	1464.84	3.70 ± 0.74
	MemoryBank [39]	1464.84	1.91 ± 0.31
	ReadAgent [17]	1464.84	120,069.68 ± 1,673.39

Findings:

- Space Complexity: All ≈ O(N); no extra storage overhead for A-MEM.
- Retrieval Time: A-MEM 0.31 → 3.70 µs (1K → 1M memories).
- MemoryBank slightly faster but less expressive; A-MEM offers richer memory representation.

Conclusion:

- A-MEM is highly scalable and efficient, handling million-scale memories with minimal delay.
- Enables long-term and cost-effective memory for LLM Agents.

Memory Analysis

Goal: Show memory organization via t-SNE

Setup: Two dialogues from LoCoMo; blue = A-MEM, red = Base Memory (w/o LG & ME).

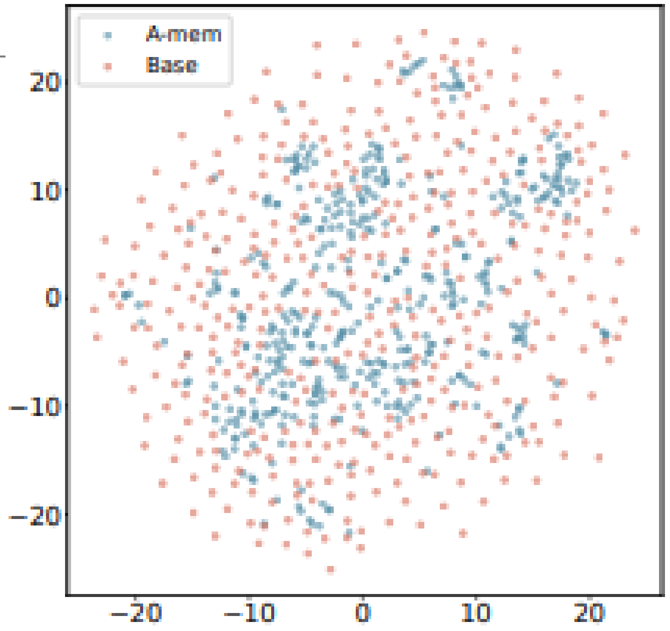
Findings:

A-MEM: clear, coherent clusters → structured and organized memory.

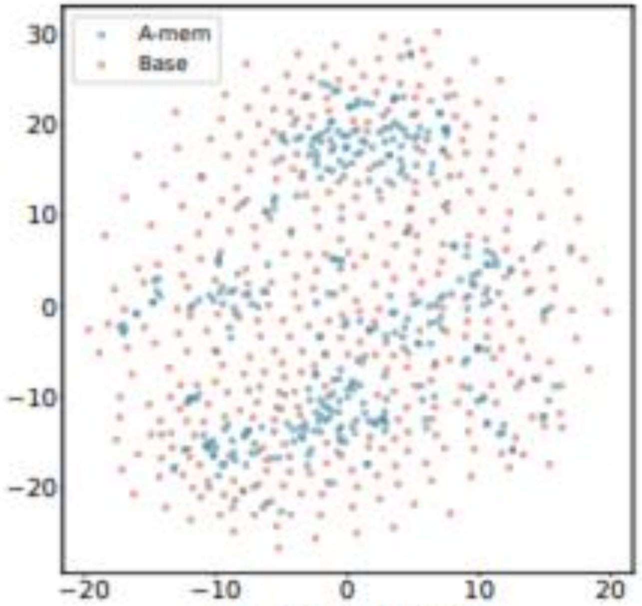
Baseline: scattered and unorganized distribution.

Conclusion:

Confirms A-MEM’s dynamic linking + evolution create well-organized, meaningful memory structures.



(a) Dialogue 1



(b) Dialogue 2

Limitation and Future Work

Limitations

- Dependent on base LLM quality (different models → different memory links).
- Currently text-only; lacks multimodal (image/audio) memory integration.
- Scalability tested up to 1M entries — but further real-world deployment yet to be explored.

Future Directions

- Extend to multimodal agentic memory.
- Improve memory quality evaluation metrics.
- Integrate with agent operating systems (e.g., AIOS) for production use.

Conclusion

- **Summary:**
  A-MEM introduces an agentic and evolving memory system that enables LLM agents to autonomously organize, link, and refine their memories.

- **Core Advantage:**
  By combining structured note-taking with dynamic linking and memory evolution, A-MEM supports long-term reasoning and adaptability.

- **Results:**
  Experiments across multiple foundation models show consistent performance gains, especially on complex multi-hop reasoning tasks, with greatly reduced token usage.

- **Impact:**
  A-MEM moves LLM agents toward lifelong learning systems capable of continuously improving their understanding over time.

References

A-Mem: Agentic Memory for LLM Agents. Xu, W., Liang, Z., Mei, K., Gao, H., Tan, J., & Zhang, Y. (2025). *arXiv preprint arXiv:2502.12110v11*.

LoCoMo Dataset. Maharana, A., Lee, D.-H., Tulyakov, S., Bansal, M., Barbieri, F., & Fang, Y. (2024). Evaluating very long-term conversational memory of llm agents. *arXiv preprint arXiv:2402.17753*.

DialSim Dataset. Kim, J., Chay, W., Hwang, H., Kyung, D., Chung, H., Cho, E., Jo, Y., & Choi, E. (2024). Dialsim: A real-time simulator for evaluating long-term multi-party dialogue understanding of conversational agents. *arXiv preprint arXiv:2406.13144*.

MemGPT. Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., & Gonzalez, J. E. (2023). Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*.

MemoryBank. Zhong, W., Guo, L., Gao, Q., Ye, H., & Wang, Y. (2024). Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

ReadAgent. Lee, K.-H., Chen, X., Furuta, H., Canny, J., & Fischer, I. (2024). A human-inspired reading agent with gist memory of very long contexts. *arXiv preprint arXiv:2402.09727*.

Sentence-BERT (all-MiniLM-L6-v2). Reimers, N., & Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*.