Language Agent Tree Search Unifies Reasoning, Acting, and Planning in Language Models

Andy Zhou [1 2] **Kai Yan** [1] **Michal Shlapentokh-Rothman** [1] **Haohan Wang** [1] **Yu-Xiong Wang** [1]

Motivation

1 Single-path methods are brittle

CoT/ReAct usually follow one autoregressive trajectory. In multi-step tasks, early errors compound and alternatives are not compared.

2 Search alone is insufficient

ToT/RAP explore multiple reasoning chains, but rely mainly on internal LM knowledge or simulated dynamics, so real feedback is underused.

3 Agents need closed-loop planning

Interactive tasks need rollback, trial-and-error, evaluation, and reusable observations/rewards from previous attempts.

Core motivation LATS extends ReAct-style interaction into MCTS: it searches over candidate trajectories using external feedback, value estimates, and self-reflection to unify reasoning, acting, and planning.

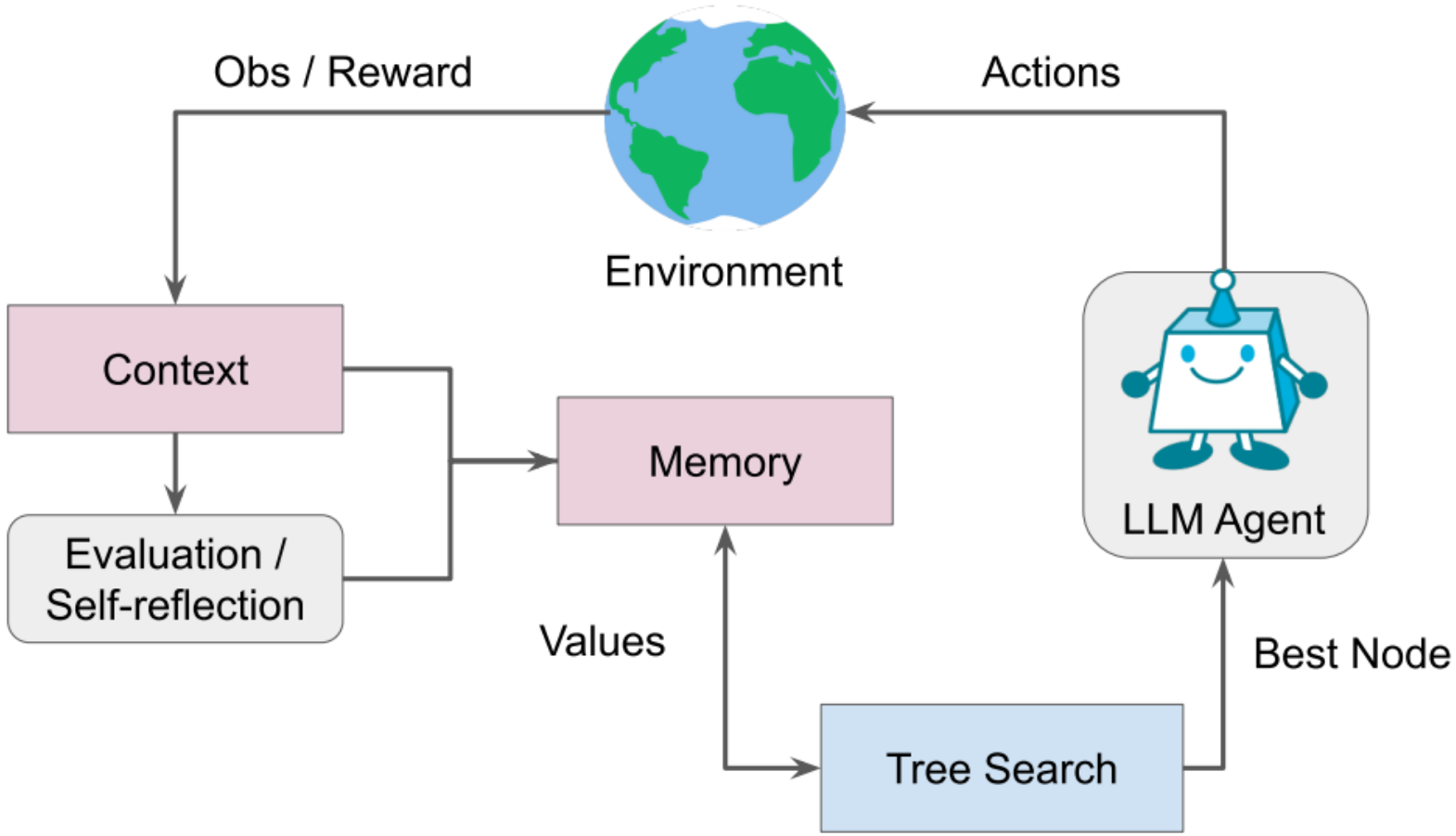

Obs / Reward
Actions
Environment
Context
Memory
LLM Agent
Evaluation /
Self-reflection
Values
Best Node
Tree Search

FIGURE EXPLANATION

Figure Explanation: LATS Overview

1

Unified agent loop

The figure shows LATS connecting an LLM agent, an external environment, memory, evaluation/self-reflection, and tree search in one loop.

2

Search over actions and reasoning

The paper states LATS expands ReAct into a search over possible reasoning and acting steps, instead of following one reflexive trajectory.

3

Feedback-guided decision making

External observations/rewards, LM-powered values, and self-reflections guide MCTS toward the best node for the agent to execute.

Approach	Reasoning	Acting	Planning	Self-Reflection	External Memory
CoT (Wei et al., 2022)	✓	×	×	×	×
ReAct (Yao et al., 2023b)	✓	✓	×	×	×
ToT (Yao et al., 2023a)	✓	×	✓	✓	✓
RAP (Hao et al., 2023)	✓	×	✓	×	✓
Self-Refine (Madaan et al., 2023)	✓	×	×	✓	×
Beam Search (Xie et al., 2023)	✓	×	×	✓	×
Reflexion (Shinn et al., 2023)	✓	✓	×	✓	✓
LATS (Ours)	✓	✓	✓	✓	✓

TABLE EXPLANATION

Table Explanation: Related Work

1

What the columns mean

The table compares reasoning, acting, planning, self-reflection, and external memory across prompting and agent methods.

2

Most prior methods cover only part of the space

ReAct supports reasoning and acting but not planning; ToT and RAP support planning but not acting.

3

Positioning of LATS

LATS is marked as supporting all five columns; the paper describes it as the first work incorporating reasoning, acting, and planning.

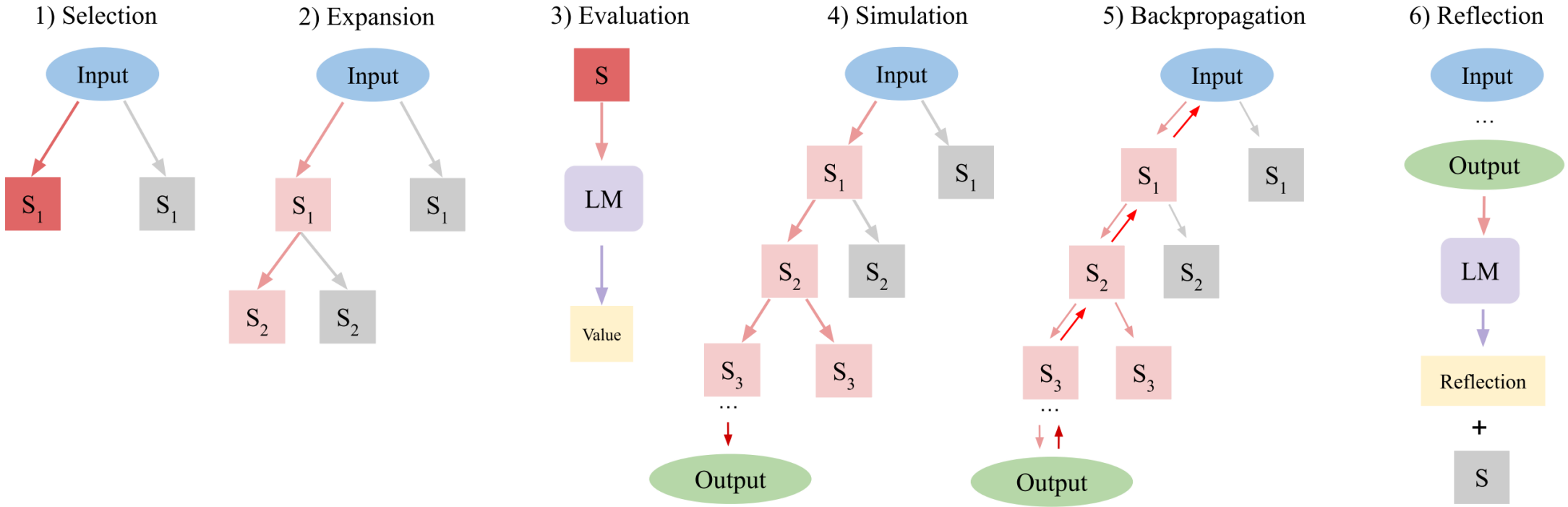

1) Selection
Input
S1
S1
2) Expansion
Input
S1
S1
S2
S2
3) Evaluation
S
LM
Value
4) Simulation
Input
S1
S1
S2
S2
S3
S3
...
Output
5) Backpropagation
Input
S1
S1
S2
S2
S3
S3
...
Output
6) Reflection
Input
...
Output
LM
Reflection
+
S

FIGURE EXPLANATION

Figure Explanation: Six LATS Operations

1

MCTS-style search cycle

LATS repeatedly performs selection, expansion, evaluation, simulation, and backpropagation until the task succeeds or the budget is reached.

2

Evaluation guides exploration

Each new child node receives a value for selection and backpropagation; the value function combines an LM score and self-consistency.

3

Failure becomes future context

If a terminal trajectory fails, LATS generates a reflection and uses it as additional context for future trials.

Experiments

Prompt Method	HotpotQA (EM) ↑
ReAct (Yao et al., 2023b)	0.32
ReAct (best of k)	0.38
Reflexion (Shinn et al., 2023)	0.51
ToT (ReAct)	0.39
RAP (ReAct)	0.54
LATS (ReAct)	0.63
LATS ($n = 3$)	0.58
LATS ($n = 10$)	0.65
LATS (CoT + ReAct)	**0.71**

Prompt Method	HotpotQA (EM) ↑
Base LM	0.32
CoT (Wei et al., 2022)	0.34
CoT - SC (Wang et al., 2022)	0.38
ToT (Yao et al., 2023a)	0.55
RAP (Hao et al., 2023)	0.60
RAP ($n = 10$)	0.60
LATS (CoT)	**0.62**

EXPERIMENT RESULTS

Results Interpretation: HotPotQA

1

Reasoning-only search improves EM

LATS (CoT) reaches 0.62 EM, above ToT 0.55 and RAP 0.60 in the reasoning-only HotPotQA setting.

2

Acting-based LATS is strongest

LATS (CoT + ReAct) reaches 0.71 EM; LATS (ReAct) reaches 0.63 EM, above ReAct, Reflexion, ToT(ReAct), and RAP(ReAct).

3

Paper's interpretation

The paper states that combining internal and external reasoning gives the highest performance, and adapting search to decision-making is non-trivial.

Prompt Method	Model	Pass@1 ↑
CoT (Wei et al., 2022)	GPT-3.5	46.9
ReAct (Yao et al., 2023b)	GPT-3.5	56.9
Reflexion (Shinn et al., 2023)	GPT-3.5	68.1
ToT (Yao et al., 2023a)	GPT-3.5	54.4
RAP (Hao et al., 2023)	GPT-3.5	63.1
LATS (ReAct)	GPT-3.5	**83.8**
Base LM	GPT-4	80.1
Reflexion	GPT-4	91.0
LATS (ReAct)	GPT-4	**92.7**

Prompt Method	Pass@1 ↑
CoT (Wei et al., 2022)	54.9
ReAct (Wei et al., 2022)	67.0
Reflexion (Shinn et al., 2023)	70.0
ToT (Yao et al., 2023a)	65.8
RAP (Hao et al., 2023)	71.4
LATS (ReAct)	**81.1**

EXPERIMENT RESULTS

Results Interpretation: Programming

1

HumanEval: highest Pass@1

LATS reaches 83.8 Pass@1 with GPT-3.5 and 92.7 with GPT-4, the best values shown in Table 4.

2

MBPP: same pattern

On MBPP, LATS reaches 81.1 Pass@1, above RAP 71.4, Reflexion 70.0, ReAct 67.0, ToT 65.8, and CoT 54.9.

3

Paper's interpretation

The paper states that both search and semantic feedback are crucial, and external feedback is important for difficult reasoning tasks such as programming.

Method	Score ↑	SR ↑
ReAct (Yao et al., 2023b)	53.8	28.0
ReAct (best of k)	59.1	32.0
Reflexion (Shinn et al., 2023)	64.2	35.0
LATS (ReAct)	**75.9**	**38.0**
IL (Yao et al., 2022)	59.9	29.1
IL+RL (Yao et al., 2022)	62.4	28.7
Fine-tuning (Furuta et al., 2024)	67.5	45.0
Expert	82.1	59.6

Prompt Method	Game of 24 (Success Rate) ↑
CoT (Wei et al., 2022)	0.08
Reflexion (Shinn et al., 2023)	0.12
ToT (Yao et al., 2023a)	0.20
RAP (Hao et al., 2023)	0.40
LATS (CoT)	**0.44**

EXPERIMENT RESULTS

Results Interpretation: WebShop / Game of 24

1

WebShop: stronger exploration

LATS obtains 75.9 score and 38.0 SR. The paper says this indicates more effective exploration for the same number of iterations.

2

Training baselines are mixed

LATS has higher score than the listed training baselines; its SR is above IL/IL+RL, but below fine-tuning and expert.

3

Game of 24: pure reasoning gain

LATS (CoT) reaches 0.44 success rate, above RAP 0.40 and ToT 0.20; the paper links this to the value function with self-consistency.

Prompt Method	HotPotQA (EM) ↑
ToT (ReAct)	0.39
RAP (ReAct)	0.54
LATS (No LM Heuristic)	0.37
LATS (DFS)	0.42
LATS (No Reflection)	0.58
LATS (ReAct)	**0.63**

EXPERIMENT RESULTS

Results Interpretation: Ablation

1

All components matter

Full LATS (ReAct) reaches 0.63 EM, the best value in Table 8.

2

Reflection helps, but is not the only factor

Removing reflection lowers EM from 0.63 to 0.58, a 0.05 drop reported in the paper.

3

Search/value design is critical

DFS drops to 0.42 and removing the LM heuristic drops to 0.37; the paper states LM scoring is crucial for leveraging external feedback.

Method	k	HotPotQA ↑	# of Nodes ↓
ToT	10	0.34	33.97
RAP	10	0.44	31.53
LATS	10	0.44	28.42
ToT	30	0.39	47.54
RAP	30	0.50	37.71
LATS	30	0.52	34.12
ToT	50	0.49	84.05
RAP	50	0.54	70.60
LATS	50	**0.61**	**66.65**

EXPERIMENT RESULTS

Results Interpretation: Cost / Efficiency

1

Higher accuracy at larger k

At k = 50, LATS reaches 0.61 HotPotQA accuracy, compared with RAP 0.54 and ToT 0.49.

2

Fewer nodes for successful search

At k = 50, LATS uses 66.65 nodes, fewer than RAP 70.60 and ToT 84.05.

3

Paper's interpretation

Across sampled trajectory budgets, the paper states LATS achieves the highest accuracy and lowest average nodes/states required for success.

AGENTSQUARE: AUTOMATIC LLM AGENT SEARCH IN MODULAR DESIGN SPACE

Yu Shang[1]*, **Yu Li**[2]*, **Keyu Zhao**[1], **Likai Ma**[1], **Jiahe Liu**[1], **Fengli Xu**[1]†, **Yong Li**[1]†
[1]Department of Electronic Engineering, Tsinghua University
[2]Shenzhen International Graduate School, Tsinghua University
{fenglixu,liyong07}@tsinghua.edu.cn

MOTIVATION

Motivation: AgentSquare

1

Manual designs do not scale

Agentic systems are often task-specific, expert-designed, and hard to adapt to novel tasks.

2

Prior search misses modular reuse

Prompt or code search does not explicitly recombine useful modules from different agent designs.

3

Evaluation cost limits search

Large module spaces and real task testing make exhaustive agent search economically hard.

Core motivation MoLAS abstracts agents into Planning, Reasoning, Tool Use, and Memory; AgentSquare searches this modular space with evolution, recombination, and prediction.

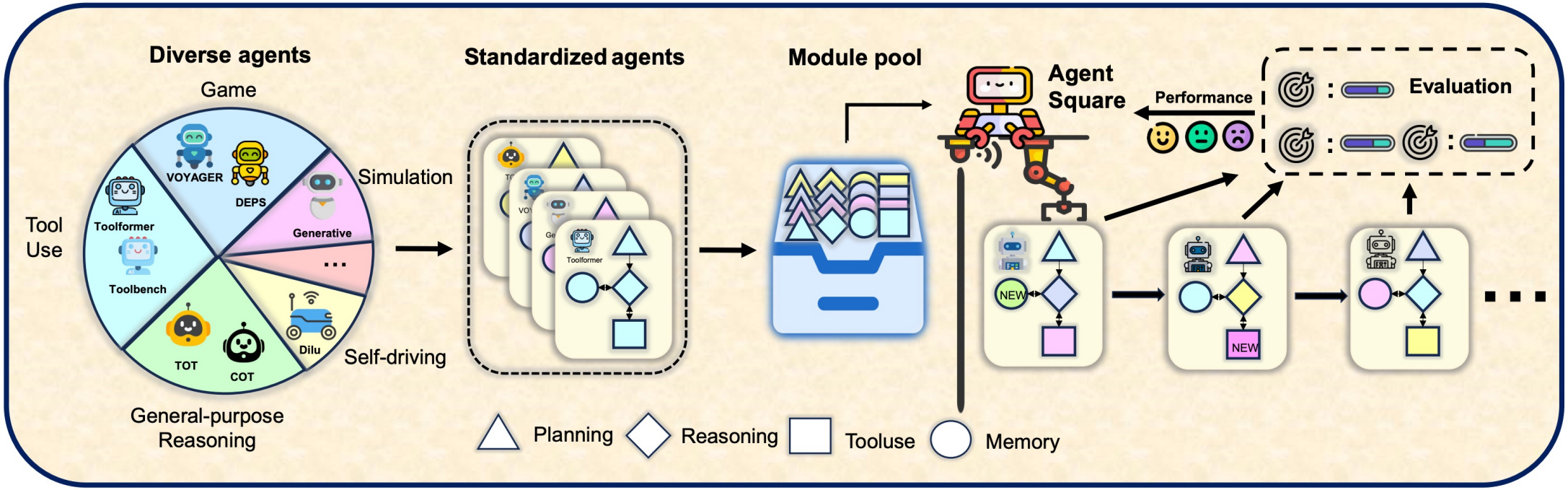

Diverse agents
Game
VOYAGER
DEPS
Simulation
Generative
...
Tool
Use
Toolformer
Toolbench
Dilu
Self-driving
TOT
COT
General-purpose
Reasoning
Standardized agents
Toolformer
Module pool
Agent
Square
Performance
Evaluation
NEW
NEW
Planning
Reasoning
Tooluse
Memory

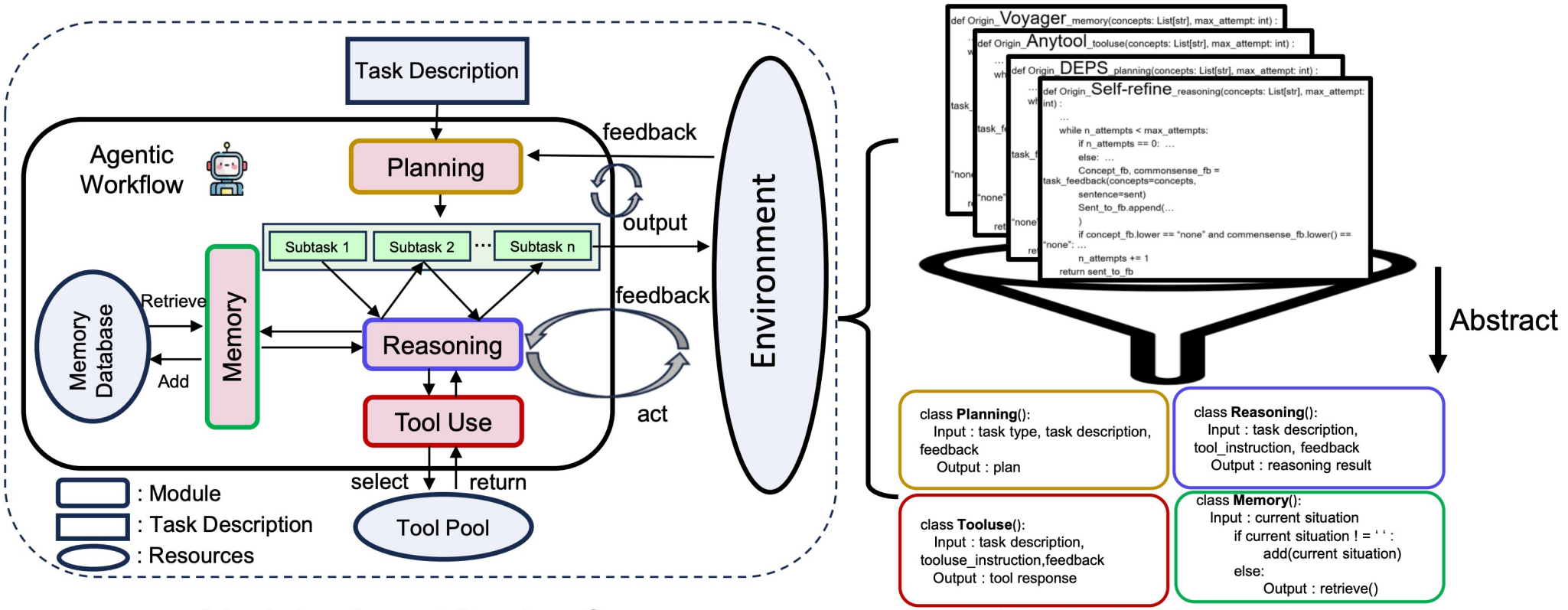
FIGURE EXPLANATION

AgentSquare Overview

1. **From diverse agents to modules**
   The figure shows task-specific agents being standardized into a reusable module pool.

2. **Four module dimensions**
   The modular design space organizes agents into Planning, Reasoning, Tool Use, and Memory.

3. **Search with feedback**
   AgentSquare evaluates candidate agents and uses the results to discover better module combinations.



Modular Agent Design Space

Standard IO Interface

FIGURE EXPLANATION

Modular Agent Design Space

1. **Iterative workflow**
   Planning decomposes tasks; reasoning solves subtasks; tools and memory support trial-and-error execution.

2. **Standard IO is central**
   Uniform input-output interfaces make modules interchangeable and extensible across prior agent designs.

3. **Concrete search space**
   The paper abstracts 16 popular agents into 1050 possible module combinations.

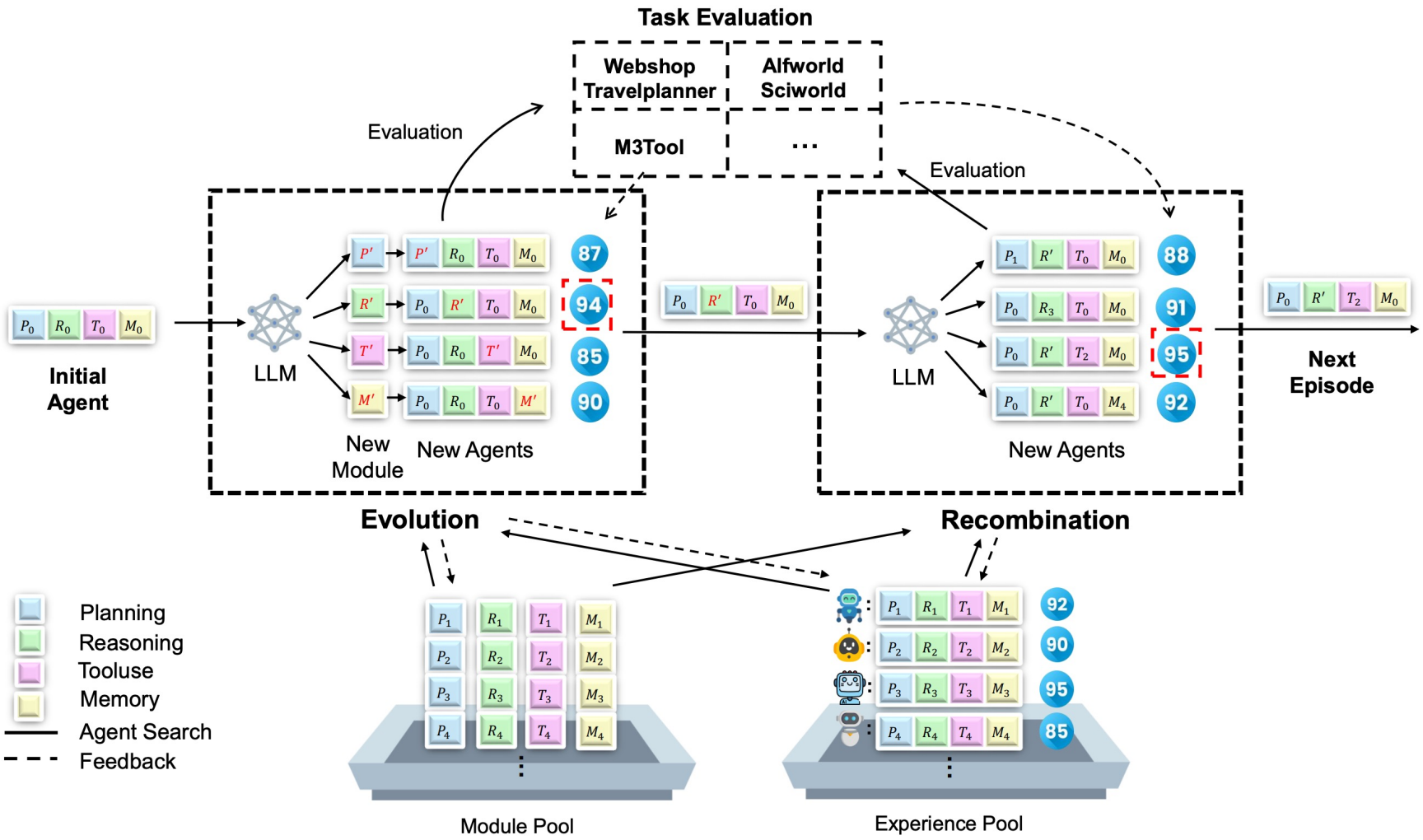

Task Evaluation
Webshop
Travelplanner
Alfworld
Sciworld
M3Tool
...
Evaluation
Evaluation
Initial Agent
LLM
New Module
New Agents
LLM
New Agents
Next Episode
Evolution
Recombination
Planning
Reasoning
Tooluse
Memory
Agent Search
Feedback
Module Pool
Experience Pool

FIGURE EXPLANATION

AgentSquare Search Framework

1. **Two search operations**
   Module recombination mixes existing modules; module evolution creates task-tailored new modules.

2. **Experience guides proposals**
   The LLM proposer uses task descriptions, module pools, and evaluated performance history.

3. **Predictor reduces cost**
   A performance predictor scores recombined agents as an in-context surrogate before real evaluation.

Experiments

		Web	Embodied		Tool		Game
Baseline Type	**Method**	**Webshop**	**ALFWorld**	**SciWorld**	**M3Tool**	**Travel**	**PDDL**
Hand-crafted Agents	CoT	0.485	0.405	0.697	0.448	0.487	0.542
	Cot-SC	0.512	0.426	0.656	0.461	0.413	0.495
	Self-refine	0.461	0.567	0.654	0.442	0.000	0.514
	ToT	0.501	0.437	0.741	0.453	0.380	0.476
	Step Back	0.468	0.279	0.220	0.434	0.000	0.486
	TP	0.398	0.404	0.576	0.387	0.430	0.518
	HuggingGPT	0.519	0.481	0.680	0.354	0.510	0.584
	Voyager	0.366	0.425	0.776	0.247	0.523	0.412
	Generative Agents	0.499	0.477	0.663	0.402	0.480	0.553
	DEPS	0.481	0.459	0.740	0.278	0.540	0.591
	OPENAGI	0.506	0.510	0.718	0.322	0.533	0.616
	Dilu	0.451	0.433	0.682	0.475	0.360	0.463
Module Search	Random	0.533	0.620	0.704	0.438	0.563	0.660
	Bayesian	0.549	0.634	0.749	0.502	0.537	0.650
Prompt Search	OPRO	0.505	0.380	0.569	0.309	0.523	0.589
Agent Search	ADAS	0.521	0.543	0.754	0.475	0.373	0.568
	AgentSquare	**0.607**	**0.695**	**0.781**	**0.524**	**0.583**	**0.669**

EXPERIMENT RESULTS

AgentSquare Main Results

1

Better than human designs

AgentSquare outperforms best-known human-designed agents across all six benchmark tasks.

2

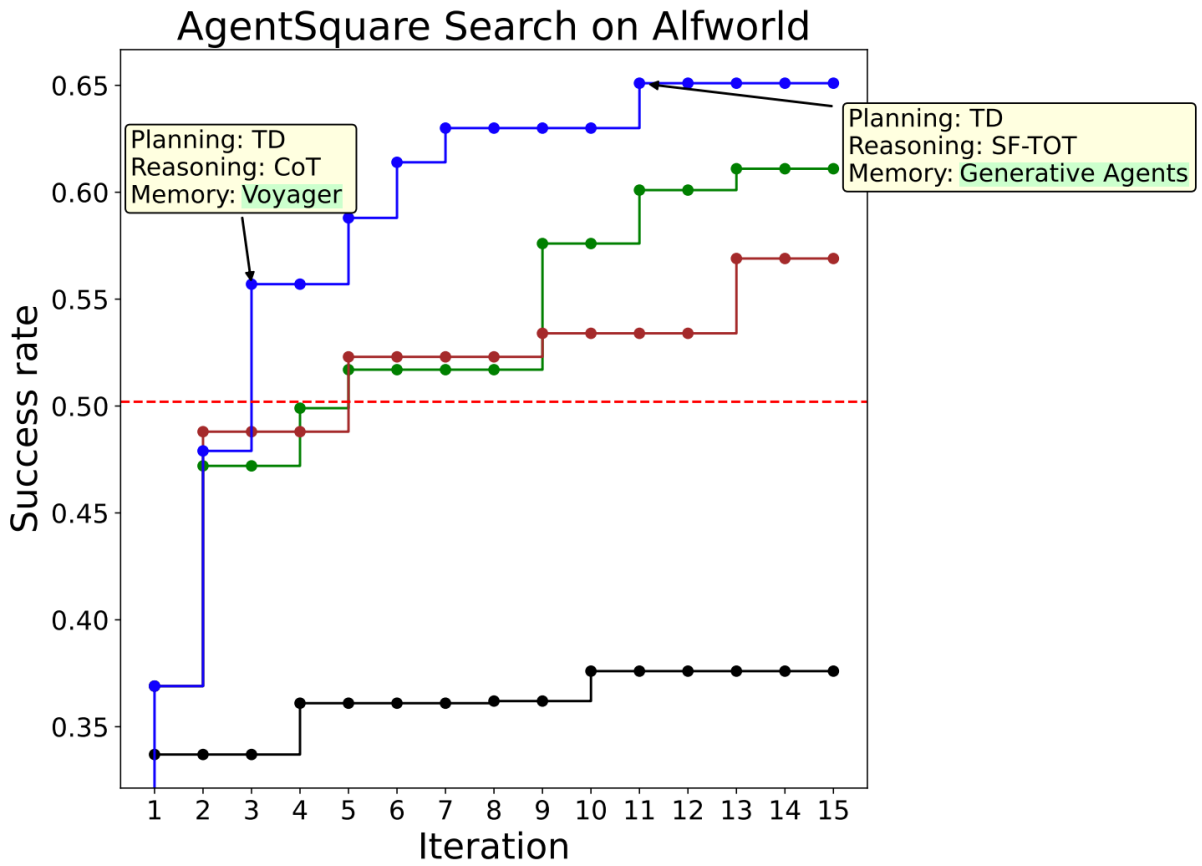
Reported task gains

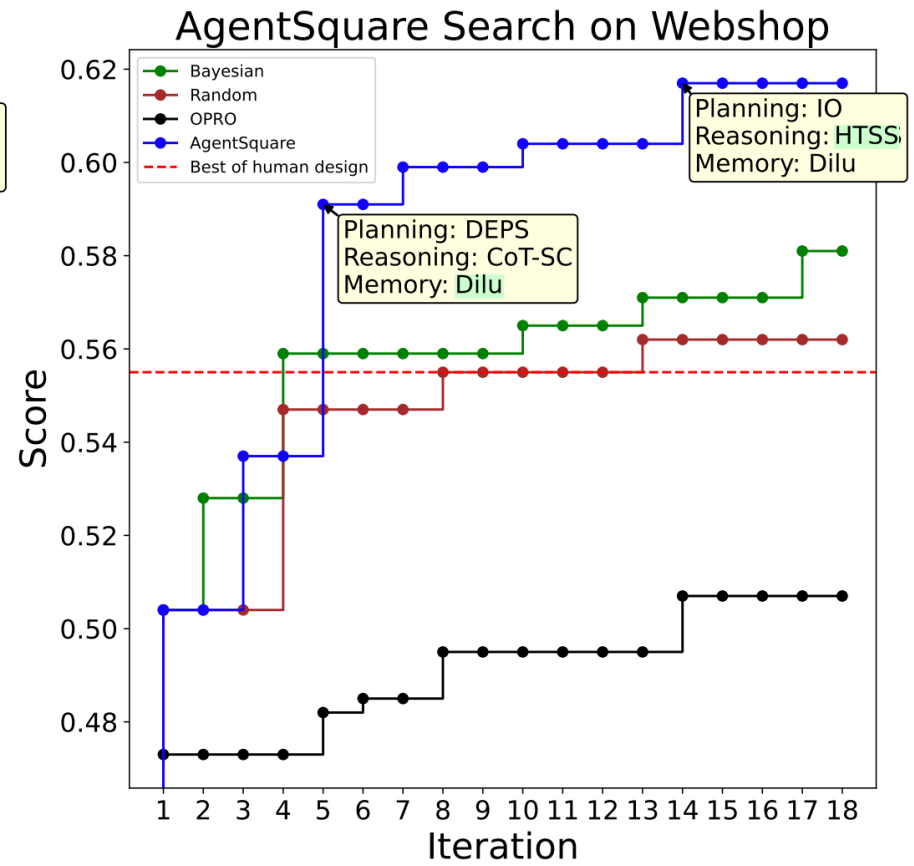
The paper reports gains of 14.1% WebShop, 26.1% ALFWorld, 20.5% SciWorld, and 30.6% M3Tool.

3

Search baselines are also lower

Under a fixed LLM token budget, AgentSquare also beats module, prompt, and agent-search baselines.


AgentSquare Search on Alfworld
Success rate
Iteration
Planning: TD
Reasoning: CoT
Memory: Voyager
Planning: TD
Reasoning: SF-TOT
Memory: Generative Agents


AgentSquare Search on Webshop
Score
Iteration
Bayesian
Random
OPRO
AgentSquare
Best of human design
Planning: DEPS
Reasoning: CoT-SC
Memory: Dilu
Planning: IO
Reasoning: HTSS
Memory: Dilu

EXPERIMENT RESULTS

Search Trajectory

1

Steady improvement

On ALFWorld and WebShop, AgentSquare shows a steady convergence trajectory over search iterations.

2

Baselines plateau

The paper says random and Bayesian search lack direction, while OPRO has limited modification space.

3

Architecture insight

Annotated points show the discovered planning, reasoning, and memory choices as performance improves.

Method	Webshop	ALFWorld	SciWorld	M3Tool	TravelPlanner	PDDL
AgentSquare (full)	**0.607**	**0.695**	**0.781**	**0.524**	**0.583**	**0.669**
w/o module evolution	0.564	0.649	0.736	0.502	0.577	0.614
w/o module recombination	0.560	0.616	0.710	0.481	0.280	0.669

EXPERIMENT RESULTS

Evolution and Recombination Ablation

1

Full model is strongest

Across six GPT-4o tasks, the full AgentSquare row is higher than removing either core operation.

2

Recombination is critical

The paper states module recombination has larger impact because it expands the search space.

3

Evolution adds task-specific modules

Module evolution helps discover advanced modules tailored to the target task.

SF-TOT

Insights: Develop a module that not only generates multiple paths and evaluates them but also incorporates self-reflection and self-improvement strategies.

```
class REASONING_SF_TOT():
 def __call__(self, ...):
  prompt = f'''Interact with a
       household to solve a task. Your
       instructions must follow the
       examples. Here are some
       examples.  ... '''
  responses = llm_response(...)
  response = self.get_votes(...)
  response = self.refine(response, ...)
  return response
```

TD

Insights: Use a timeline-based approach, where tasks are decomposed into sub-tasks with explicit temporal dependencies.

```
class PLANNING_TD():
 def __call__(self, ...):
  prompt = f'''You are a planner who
       divides a {task_type} task into
       several subtasks with explicit
       temporal dependencies. Consider
       the order of actions and their
       dependencies to ensure logical
       sequencing.  ...'''
  sub_plans = llm_response(...)
  return sub_plans
```

FIGURE EXPLANATION

Discovered ALFWorld Modules

1

Best agent mixes old and new

The paper's ALFWorld example combines Generative Agents memory with newly discovered TD and SF-ToT modules.

2

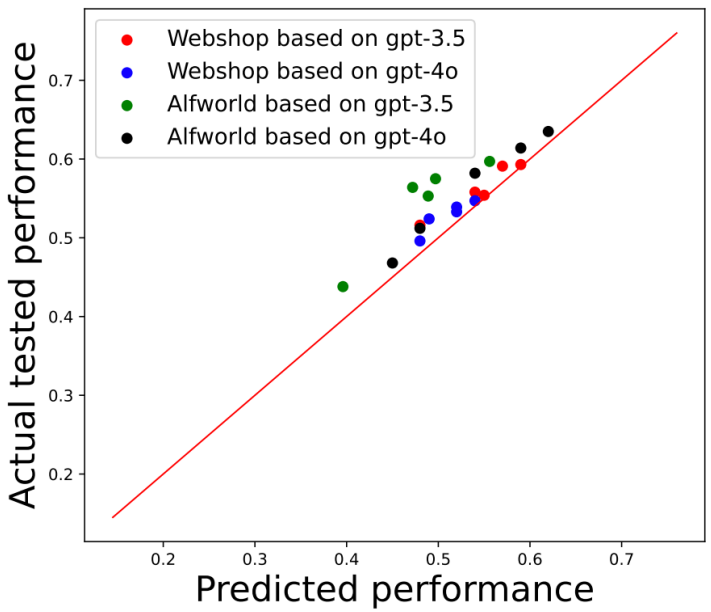
TD planning

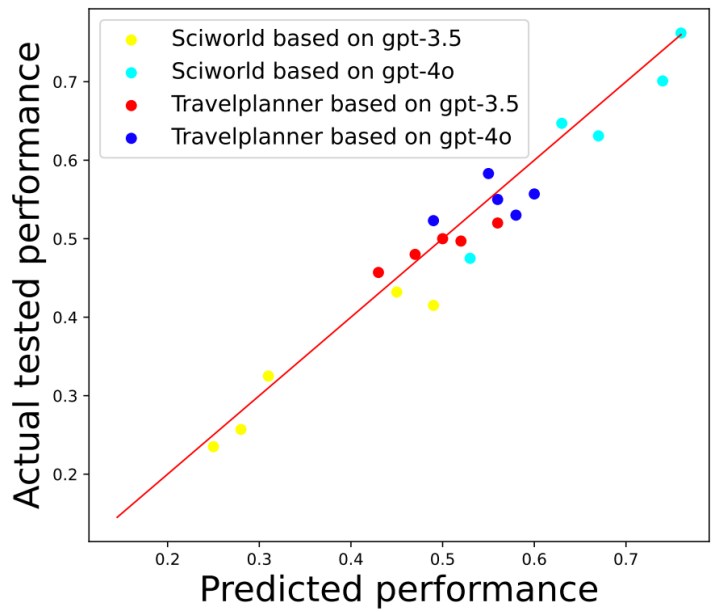
TD decomposes tasks into subtasks with explicit temporal dependencies.

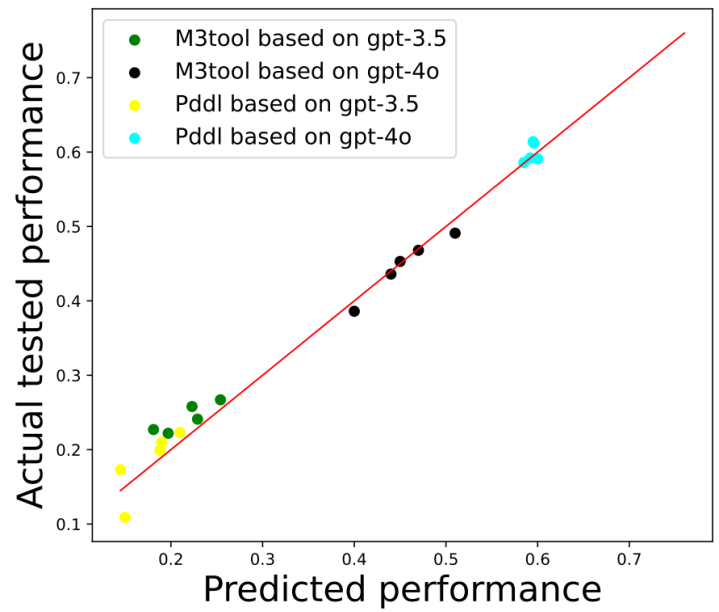
3

SF-ToT reasoning

SF-ToT generates and evaluates multiple paths, then adds self-reflection and self-improvement.


Webshop based on gpt-3.5
Webshop based on gpt-4o
Alfworld based on gpt-3.5
Alfworld based on gpt-4o
Actual tested performance
Predicted performance
0.2
0.3
0.4
0.5
0.6
0.7


Sciworld based on gpt-3.5
Sciworld based on gpt-4o
Travelplanner based on gpt-3.5
Travelplanner based on gpt-4o
Actual tested performance
Predicted performance
0.2
0.3
0.4
0.5
0.6
0.7


M3tool based on gpt-3.5
M3tool based on gpt-4o
Pddl based on gpt-3.5
Pddl based on gpt-4o
Actual tested performance
Predicted performance
0.1
0.2
0.3
0.4
0.5
0.6
0.7

EXPERIMENT RESULTS

Performance Predictor Validation

1

Predictions track reality

Predicted performance closely aligns with actual tested performance across tasks and GPT-3.5/GPT-4o.

2

Used for recombination

The predictor serves as an in-context surrogate for evaluating newly recombined agents.

3

Cost motivation

For ALFWorld with GPT-4o, predictor evaluation costs about 0.025% of a full evaluation.

Method	Webshop	ALFWorld	SciWorld	M3Tool	TravelPlanner	PDDL
AgentSquare(full)	**0.617**	**0.651**	**0.432**	**0.285**	**0.520**	**0.219**
w/o module evolution	0.595	0.623	0.288	0.236	0.483	0.202
w/o module recombination	0.578	0.546	0.310	0.258	0.267	0.173

Table A.5: Ablation study of AgentSquare on GPT-3.5 on six tasks across different domains.

	Webshop	ALFWorld	SciWorld	M3Tool	TravelPlanner	PDDL
Avg cost (GPT-3.5)	$3.16	$4.25	$1.92	$2.43	$1.84	$2.70
Iterations (GPT-3.5)	23	21	8	14	9	17
Avg cost (GPT-4o)	$10.51	$13.96	$42.14	$26.03	$29.75	$26.94
Iterations (GPT-4o)	18	15	9	18	8	12

EXPERIMENT RESULTS

GPT-3.5 Ablation and Search Cost

1

Same ablation pattern

The GPT-3.5 appendix table again shows the full AgentSquare variant above both ablated variants.

2

Costs vary by domain

Table A.6 reports per-iteration costs and termination iterations across WebShop, ALFWorld, SciWorld, M3Tool, TravelPlanner, and PDDL.

3

Why prediction matters

High GPT-4o per-iteration costs motivate skipping unpromising candidates during search.

MULTI-AGENT DESIGN: OPTIMIZING AGENTS WITH BETTER PROMPTS AND TOPOLOGIES

Han Zhou[1,3,*] **Xingchen Wan**[2] **Ruoxi Sun**[2] **Hamid Palangi**[1] **Shariq Iqbal**[2]
Ivan Vulić[2,3] **Anna Korhonen**[3] **Sercan Ö. Arık**[1]
[1]Google [2]Google DeepMind [3]University of Cambridge
{hzhouml,soarik}@google.com

MOTIVATION

Motivation: Multi-Agent Design

1

MAS design is coupled

Prompts define each agent's role, while topologies determine how agents interact and collaborate.

2

Prompt sensitivity compounds

In cascaded MAS, poor prompts can propagate errors, and direct prompt optimization becomes difficult.

3

Topology search alone is insufficient

The paper finds prompts are influential and beneficial topologies are only a small search-space fraction.

Core motivation

MASS automates MAS design by interleaving local prompt optimization, pruned topology optimization, and global workflow prompt optimization.

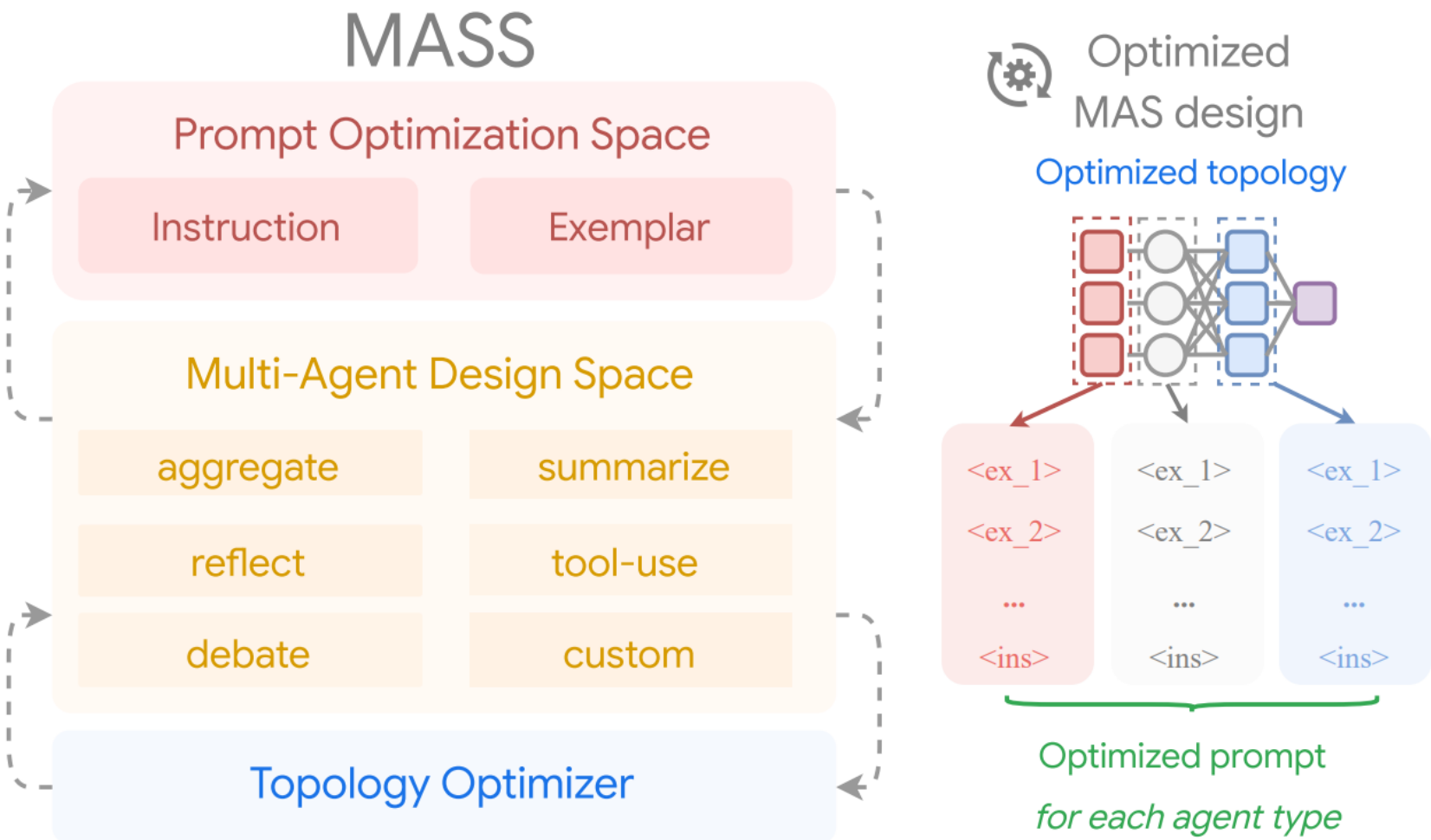

MASS
Prompt Optimization Space
Instruction
Exemplar
Multi-Agent Design Space
aggregate
summarize
reflect
tool-use
debate
custom
Topology Optimizer
Optimized
MAS design
Optimized topology
<ex_1>
<ex_2>
...
<ins>
<ex_1>
<ex_2>
...
<ins>
<ex_1>
<ex_2>
...
<ins>
Optimized prompt
for each agent type

FIGURE EXPLANATION

MASS Framework

1. **Two spaces optimized**
   MASS searches both prompt optimization space and multi-agent design space.

2. **Output is a complete MAS**
   The optimized design includes both an effective topology and optimized prompts for each agent type.

3. **Customizable design space**
   The framework supports agent types such as aggregate, reflect, debate, summarize, tool-use, and custom agents.

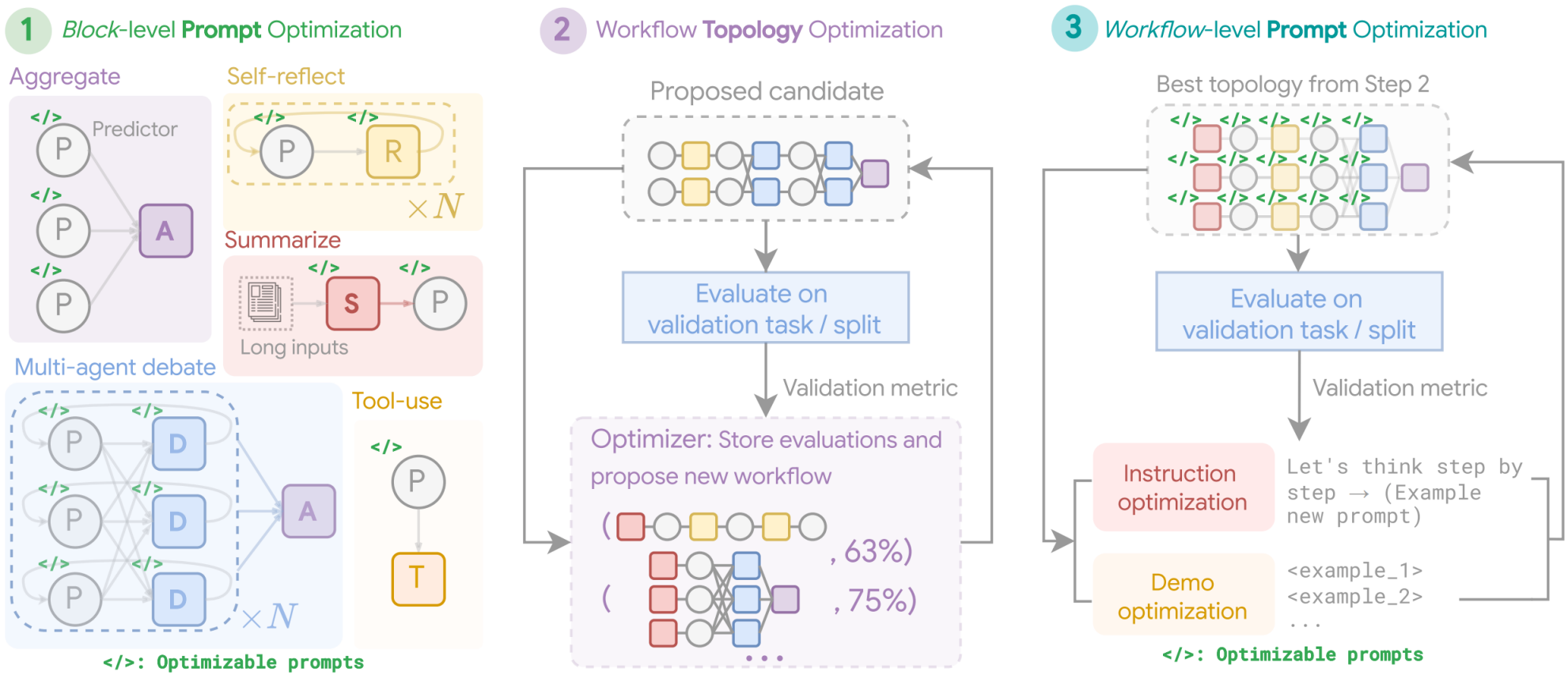

1 Block-level Prompt Optimization
Aggregate
Predictor
Self-reflect
Summarize
Long inputs
Multi-agent debate
Tool-use
×N
</>: Optimizable prompts
2 Workflow Topology Optimization
Proposed candidate
Evaluate on validation task / split
Validation metric
Optimizer: Store evaluations and propose new workflow
, 63%)
, 75%)
3 Workflow-level Prompt Optimization
Best topology from Step 2
Evaluate on validation task / split
Validation metric
Instruction optimization
Let's think step by step → (Example new prompt)
Demo optimization
<example_1>
<example_2>
...
</>: Optimizable prompts

FIGURE EXPLANATION

Three-Stage MASS Search

1. **Prompt and topology variables**
   The search space includes instructions, exemplars, and configurable agentic building blocks.

2. **Local to global**
   MASS first optimizes each block, then topology, then the whole workflow's prompts.

3. **Conditioned stages**
   Each later stage uses the prompts or topology found by earlier stages.

Algorithm 1 MASS: Multi-Agent System Search

Input: Agentic modules in the search space $a_i \in \mathcal{A}$, workflow of agents $\mathcal{W}(a)$, prompt optimizer $\mathcal{O}$, evaluator $\mathcal{E}$, validation set $\mathcal{D}$, temperature t, number of candidates N, budget B.
Output: Optimized multi-agent system $\mathcal{W}^*$.
[**1PO**: *Block*-level **Prompt** Optimization]
Prompt optimization for the initial agent: $a_0^* \leftarrow \mathcal{O}_{\mathcal{D}}(a_0)$.
for a_i in $\mathcal{A} \setminus \{a_0\}$ **do**
    Local prompt optimization for each building block in the design space: $a_i^* \leftarrow \mathcal{O}_{\mathcal{D}}(a_i|a_0^*)$.
    Obtain incremental Influence: $I_{a_i} \leftarrow \mathcal{E}(a_i^*)/\mathcal{E}(a_0^*)$.
end for
[**2TO**: Workflow **Topology** Optimization]
Obtain the selection probability $p_a \leftarrow \mathrm{Softmax}(I_a, t)$.
while $n < N$ **do**
    Search space pruning: $\mathcal{A}_p = \{a_i\}$ for a_i in $\mathcal{A}$ if $u < p_{a_i}$, where $u \sim \mathrm{Uniform}(0, 1)$.
    Reject sampling for agentic configurations in budget: $a \sim \mathcal{A}_p$ s.t. $\mathcal{N}(a) < B$.
    Build the workflow $\mathcal{W}_c \leftarrow (a_i^*(\cdot), a_{i+1}^*(\cdot), \ldots)$ in a rule-base order with optimized prompts.
    Evaluate and record the score $\mathcal{E}_{\mathcal{D}}(\mathcal{W}_c)$.
end while
Obtain the best-performing workflow $\mathcal{W}_c^* \leftarrow \arg\max_{c \in \mathcal{C}} \mathcal{E}_{\mathcal{D}}(\mathcal{W}_c)$.
[**3PO**: *Workflow*-level **Prompt** Optimization]
Workflow-level prompt optimization for the best-performing topology: $\mathcal{W}^* \leftarrow \mathcal{O}_{\mathcal{D}}(\mathcal{W}_c^*)$.
Return optimized multi-agent system $\mathcal{W}^*$.

ALGORITHM EXPLANATION

MASS Algorithm

1

Inputs define the search

The algorithm takes modules, workflows, prompt optimizer, evaluator, validation set, candidates, and budget.

2

Pruning by influence

It computes incremental influence and uses selection probabilities to prune topology dimensions.

3

Final workflow optimization

After choosing the best workflow, MASS runs workflow-level prompt optimization and returns the optimized MAS.

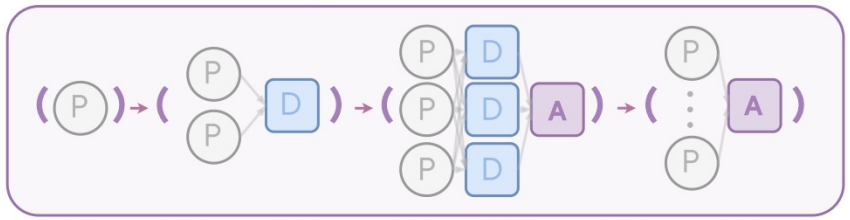
1 *Block*-level **Prompt** Optimization **(62% → 79%)**

```
Debator:
You are a seasoned math professor specializing in clear and concise
explanations.  You are reviewing student solutions to math problems.
Below, you will find the problem, followed by solutions from several
students.  Carefully examine each student's solution, identifying any
errors in their logic or calculations. Provide a comprehensive
rationale explaining your analysis of each student's work, clearly
stating whether their final answer is correct or incorrect and why.
Finally, provide your own definitive and simplified solution to the
problem, ensuring its accuracy and clarity. Present your final answer
bracketed between <answer> and </answer> at the end.

Question: Compute $17^{-1}\\pmod{83}$.
Solutions: Agent 0: 44\nAgent 1: 74
Rationale: <Rationale>
Answer: 44

<Task Demo: Exemplar_2>
<Task Demo: Examplar_3>
```

2 Workflow **Topology** Optimization **(79% → 83%)**



3 *Workflow*-level **Prompt** Optimization **(83% → 85%)**

```
Predictor:
Let's think step by step to solve the
given problem. Clearly explain your
reasoning process, showing all
intermediate calculations and
justifications.  Express your final
answer as a single numerical value or
simplified expression enclosed within
<answer></answer> tags.  Avoid
extraneous text or explanations outside
of the core reasoning and final answer.
<Task Demo: Exemplar_1>
```

FIGURE EXPLANATION

MATH Optimization Trajectory

1

Stage 1 finds debate

Starting from zero-shot CoT, block-level optimization identifies debate as a strong topology.

2

Stage 2 changes the topology

Workflow topology optimization finds that aggregation with more parallel agents can outperform debate.

3

Stage 3 adapts prompts

Workflow-level prompt optimization discovers the best prompt conditioned on the selected topology.

Experiments

Gemini-1.5-pro-002									
Task	**Reasoning**		**Multi-hop Long-context**			**Coding**			
Method	MATH	DROP	HotpotQA	MuSiQue	2WikiMQA	MBPP	HumanEval	LCB	Avg.
CoT	$71.67_{3.30}$	$70.59_{1.67}$	$57.43_{0.52}$	$37.81_{1.43}$	$63.39_{1.12}$	$68.33_{0.47}$	$86.67_{0.94}$	$66.33_{0.62}$	65.28
Self-Consistency	$77.33_{1.25}$	$74.06_{0.90}$	$58.60_{2.19}$	$41.81_{1.00}$	$67.79_{1.19}$	$69.50_{0.71}$	$86.00_{0.82}$	$70.33_{0.94}$	68.18
Self-Refine	$79.67_{2.36}$	$71.03_{1.31}$	$60.62_{3.33}$	$42.15_{1.34}$	$66.74_{2.43}$	$63.67_{0.24}$	$84.00_{1.63}$	$67.33_{1.31}$	66.90
Multi-Agent Debate	$78.67_{0.94}$	$71.78_{0.71}$	$64.87_{0.23}$	$46.00_{0.80}$	$71.78_{0.63}$	$68.67_{0.85}$	$86.67_{1.25}$	$73.67_{1.65}$	70.26
ADAS	$80.00_{0.82}$	$72.96_{0.90}$	$65.88_{1.29}$	$41.95_{1.24}$	$71.14_{0.66}$	$73.00_{1.08}$	$87.67_{1.70}$	$65.17_{1.25}$	69.72
AFlow*	$76.00_{0.82}$	$88.92_{0.63}$	$68.62_{0.47}$	$32.05_{1.29}$	$\mathbf{76.51}_{1.05}$	-	$88.00_{0.00}$	-	-
MASS (Ours)	$\mathbf{84.67}_{0.47}$	$\mathbf{90.52}_{0.64}$	$\mathbf{69.91}_{1.11}$	$\mathbf{51.40}_{0.42}$	$73.34_{0.67}$	$\mathbf{86.50}_{0.41}$	$\mathbf{91.67}_{0.47}$	$\mathbf{82.33}_{0.85}$	**78.79**
Gemini-1.5-flash-002									
CoT	$66.67_{2.36}$	$71.79_{0.69}$	$57.82_{1.10}$	$37.10_{1.35}$	$63.40_{0.68}$	$63.33_{1.25}$	$75.67_{1.89}$	$51.17_{0.24}$	60.87
Self-Consistency	$69.33_{1.25}$	$73.42_{0.19}$	$60.19_{1.01}$	$41.94_{0.93}$	$67.98_{0.72}$	$63.67_{0.62}$	$77.67_{1.89}$	$53.83_{1.18}$	63.50
Self-Refine	$71.33_{0.94}$	$73.71_{1.09}$	$58.84_{3.04}$	$41.21_{1.99}$	$65.56_{1.57}$	$63.33_{1.25}$	$81.67_{1.89}$	$52.00_{1.41}$	63.46
Multi-Agent Debate	$71.67_{0.94}$	$74.79_{0.87}$	$64.17_{1.69}$	$46.27_{1.33}$	$72.19_{0.54}$	$63.00_{0.71}$	$79.67_{1.25}$	$55.50_{0.41}$	65.91
ADAS	$68.00_{1.41}$	$75.95_{1.18}$	$61.36_{2.89}$	$\mathbf{48.81}_{1.03}$	$66.90_{1.00}$	$65.83_{0.24}$	$80.67_{2.49}$	$50.50_{1.63}$	64.75
MASS (Ours)	$\mathbf{81.00}_{2.45}$	$\mathbf{91.68}_{0.14}$	$\mathbf{66.53}_{0.38}$	$43.67_{1.21}$	$\mathbf{76.69}_{0.50}$	$\mathbf{78.00}_{0.82}$	$\mathbf{84.67}_{0.47}$	$\mathbf{72.17}_{0.85}$	**74.30**

EXPERIMENT RESULTS

MASS Main Results

1

Best average performance

MASS reports the best average results on Gemini 1.5 Pro and Gemini 1.5 Flash.

2

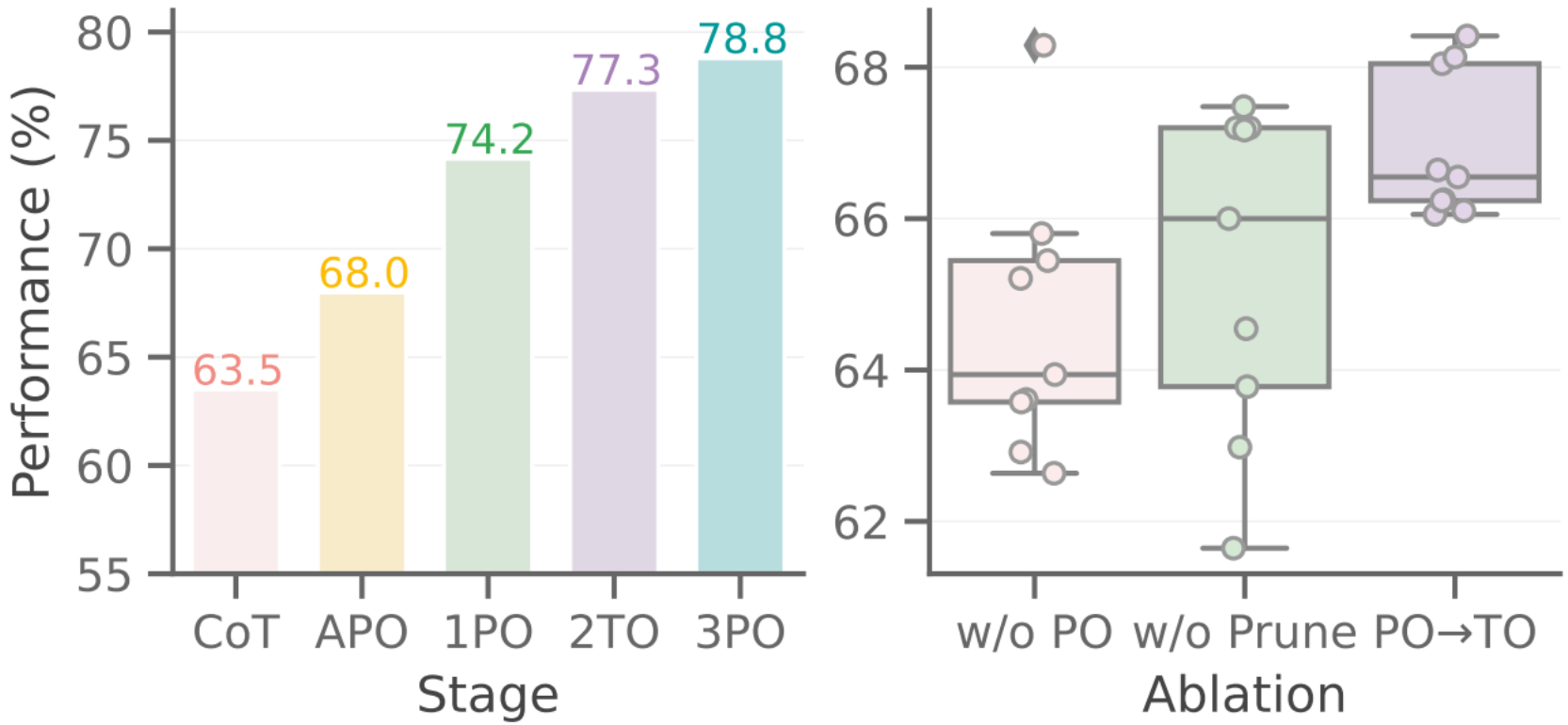
Broad task coverage

The table covers reasoning, multi-hop long-context understanding, and coding tasks with task-specific metrics.

3

Why it matters

The paper attributes the gains to searching both prompt and topology design spaces.


Performance (%)
80
75
70
65
60
55
63.5
68.0
74.2
77.3
78.8
CoT
APO
1PO
2TO
3PO
Stage
68
66
64
62
w/o PO
w/o Prune
PO→TO
Ablation

EXPERIMENT RESULTS

Stage-Wise Ablation

1

Each stage adds value

The paper reports about +6% from 1PO over single-agent APO, +3% from 2TO, and about +2% from 3PO.

2

Pruning matters

The right-side ablation shows weaker HotpotQA results without prompt optimization or without pruning.

3

Supports local-to-global design

The ablation backs the staged strategy: optimize blocks first, then topology, then workflow prompts.

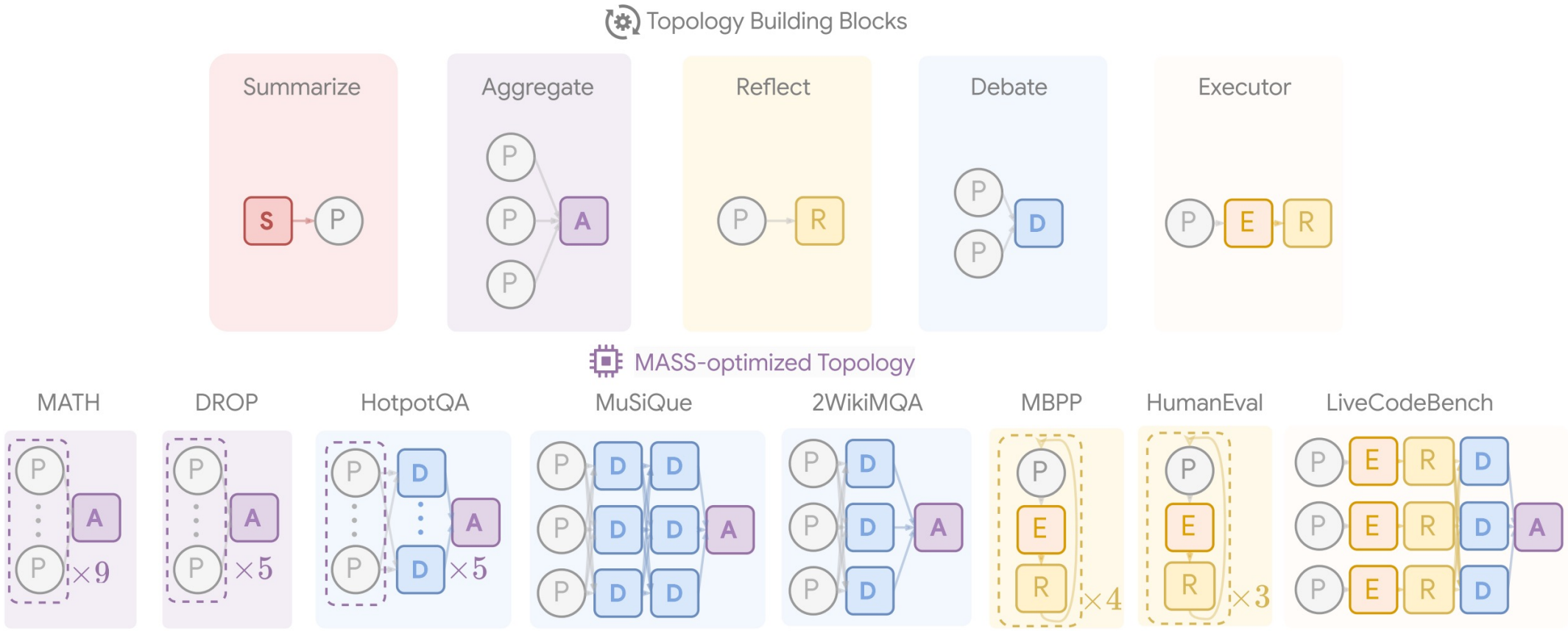

Topology Building Blocks
Summarize
S
P
Aggregate
P
P
P
A
Reflect
P
R
Debate
P
P
D
Executor
P
E
R
MASS-optimized Topology
MATH
DROP
HotpotQA
MuSiQue
2WikiMQA
MBPP
HumanEval
LiveCodeBench
×9
×5
×5
×4
×3

FIGURE EXPLANATION

Discovered Topologies

1

Task families differ

The paper observes debate helps multi-hop factual tasks, while self-consistency helps reasoning tasks.

2

Coding pattern

Coding tasks tend to share a reflection plus tool-use pattern.

3

Automatic search is still needed

Even within the same task family, best configurations differ, motivating automatic topology selection.

MAS-GPT: Training LLMs to Build LLM-based Multi-Agent Systems

Rui Ye [1 2 * @] **Shuo Tang** [1 *] **Rui Ge** [1] **Yaxin Du** [1] **Zhenfei Yin** [3 4] **Siheng Chen** [1] **Jing Shao** [2]

MOTIVATION

Motivation: MAS-GPT

1

Manual MAS lack adaptability

Fixed collaboration structures and prompts require high effort and do not adapt to each query.

2

Adaptive search is expensive

Existing optimization methods often need multiple LLM calls and a validation set for each task.

3

MAS generation needs data

LLMs have limited knowledge of generating executable MAS, so query-MAS pairs must be constructed.

Core motivation

MAS-GPT reframes building MAS as generation: given a query, output executable MAS code in one inference after SFT on consistency-oriented query-MAS pairs.

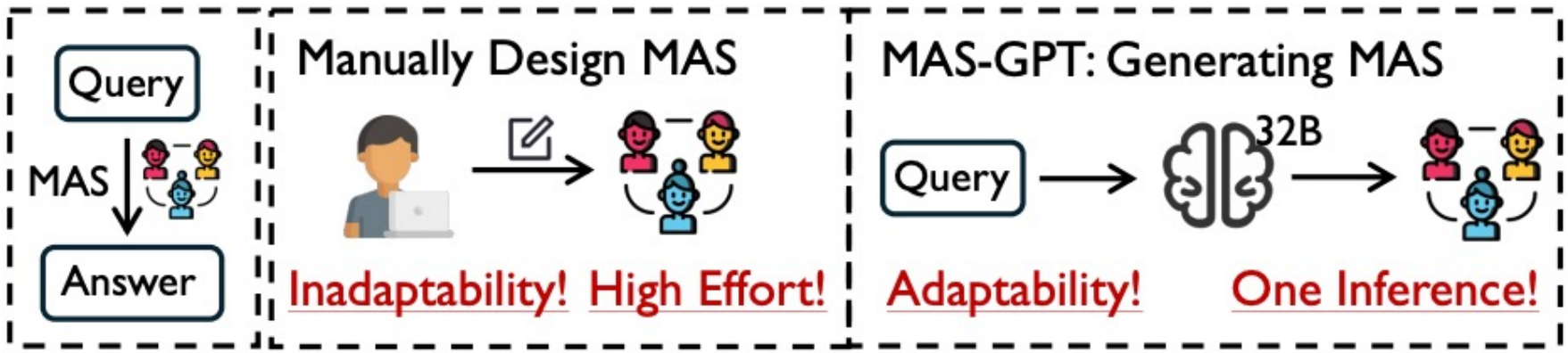

Query
MAS
Answer
Manually Design MAS
Inadaptability!
High Effort!
MAS-GPT: Generating MAS
Query
32B
Adaptability!
One Inference!

FIGURE EXPLANATION

MAS-GPT Paradigm

1

Query-specific MAS

MAS-GPT takes a query and generates an executable MAS tailored to that query.

2

One inference

The figure contrasts manual MAS design with MAS-GPT generating MAS in a single LLM inference.

3

Main benefit

The paper frames this as improving adaptability while reducing human effort and inference cost.

```
def forward(query):
    math_agent = f'You are a math expert. Solve this\
        question: {query}'
    math_output = call_llm(math_agent)

    feedback_agent = f'Given {query} and {math_output},\
        provide feedback'
    feedback_output = call_llm(feedback_agent)

    refine_agent = f'Given {query}, {math_output} and \
        {feedback_output}, provide the final answer'

    return call_llm(refine_agent)
```

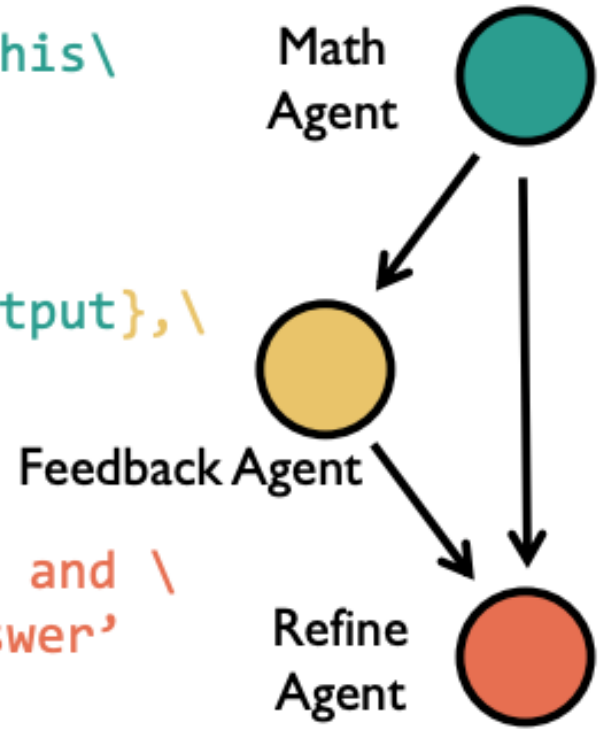


FIGURE EXPLANATION

Unified MAS Code Representation

1

Forward function

The paper represents an MAS as a Python forward function that takes a query and returns an answer.

2

Agents as variables

Agent prompts are variables, LLM calls are function calls, and interactions are string concatenations.

3

Executable output

This representation makes generated MAS directly usable for processing the query.

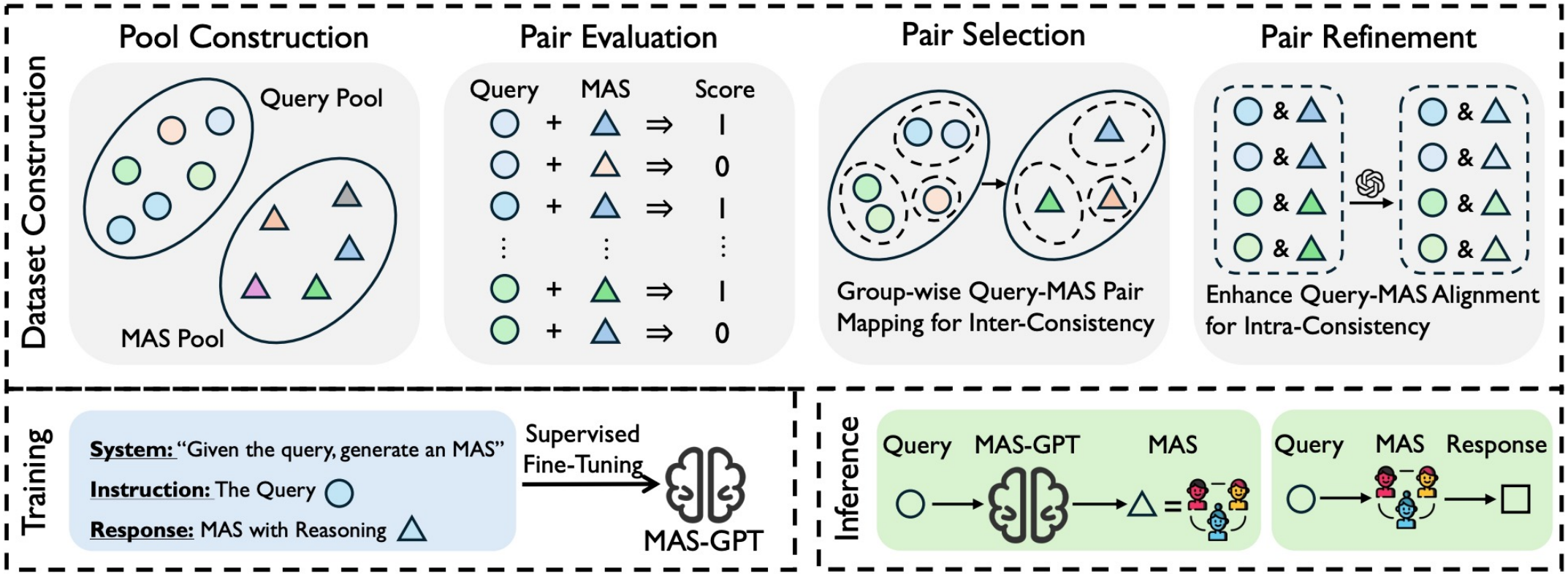

Dataset Construction
Pool Construction
Query Pool
MAS Pool
Pair Evaluation
Query
MAS
Score
1
0
1
1
0
Pair Selection
Group-wise Query-MAS Pair Mapping for Inter-Consistency
Pair Refinement
Enhance Query-MAS Alignment for Intra-Consistency
Training
System: "Given the query, generate an MAS"
Instruction: The Query
Response: MAS with Reasoning
Supervised Fine-Tuning
MAS-GPT
Inference
Query
MAS-GPT
MAS
Query
MAS
Response

FIGURE EXPLANATION

Data Construction and Training

1

Four construction steps

The pipeline builds query/MAS pools, evaluates pairs, selects pairs, and refines them before SFT.

2

Inter-consistency

Similar queries are paired with similar high-performing MAS so the model can learn generalizable patterns.

3

Intra-consistency

Refinement strengthens the relation between each query and its MAS before training.

Experiments

METHOD	MATH	GSM8K	GSM-H	H-EVAL*	H-EVAL+*	MMLU	GPQA*	SCIBENCH*	AVG.
SINGLE (DUBEY ET AL., 2024)	50.55	92.38	45.80	79.01	75.78	77.37	36.68	21.05	59.83
CHAIN-OF-THOUGHT (WEI ET AL., 2022)	53.20	92.79	46.20	77.16	77.02	75.56	35.28	17.68	59.36
SELF-CONSISTENCY (WANG ET AL., 2024B)	61.59	94.99	47.20	77.78	75.78	78.18	37.15	20.00	61.58
LLM-DEBATE (DU ET AL., 2024)	61.37	91.58	44.60	74.69	74.53	77.78	34.35	19.79	59.84
SELF-REFINE (MADAAN ET AL., 2024)	58.50	90.78	37.80	67.90	62.73	74.75	38.32	20.00	56.35
QUALITY-DIVERSITY (LU ET AL., 2024)	60.49	92.99	45.60	70.99	70.19	75.76	33.64	20.63	58.79
SPP (WANG ET AL., 2024C)	51.66	92.79	44.80	76.54	73.29	77.37	35.05	20.84	59.04
AGENTVERSE (CHEN ET AL., 2024)	55.63	93.39	41.40	77.78	73.91	76.57	40.19	16.00	59.36
GPTSWARM (ZHUGE ET AL., 2024)	55.41	93.19	43.20	69.14	73.91	75.15	36.45	14.11	57.57
DYLAN (LIU ET AL., 2024B)	59.60	91.18	44.80	79.01	75.78	78.18	35.98	19.79	60.54
MAS-GPT (OURS)	68.65	93.39	62.40	80.25	78.88	78.38	37.62	24.21	**65.47**

EXPERIMENT RESULTS

MAS-GPT Main Results

1

Best average over baselines

Using Llama-3-70B-Instruct as MAS driver, MAS-GPT is best on average over 10 baselines.

2

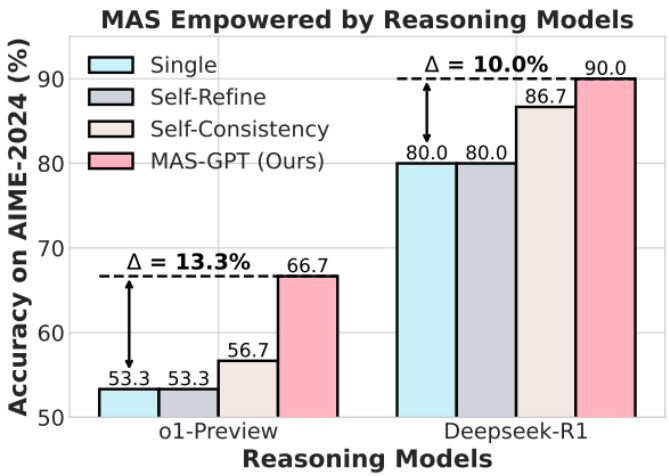
Reported margin

The paper states MAS-GPT outperforms the second-best method by 3.89% on average.

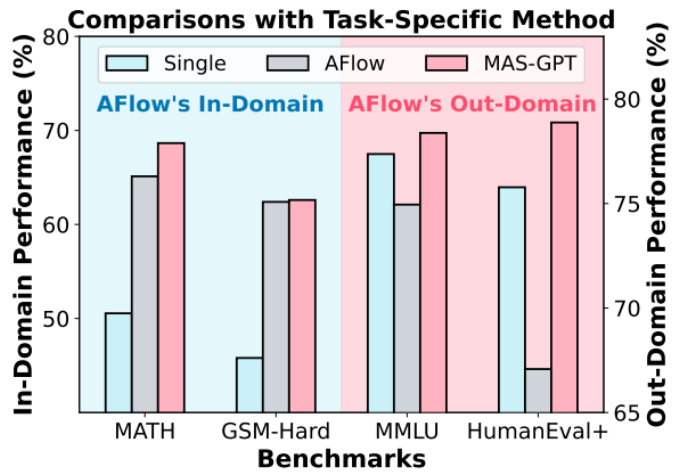
3

Generalization evidence

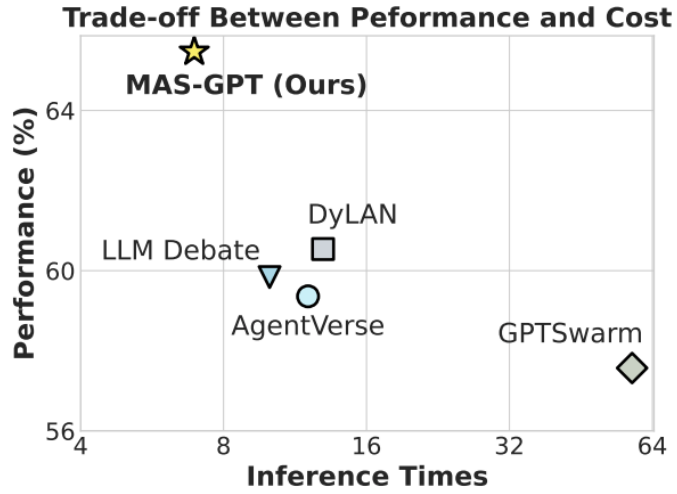
It performs well on both in-domain and out-of-domain benchmarks and is consistently above single agent.



(a) MAS-GPT-Assisted Reasoning



(b) Comparison with Task-Specific Method



(c) Performance v.s. Inference Times

EXPERIMENT RESULTS

Reasoning and Cost Comparisons

1

Stronger reasoning LLMs improve

With o1-preview, MAS-GPT improves over a single LLM by 13.34% in the paper's Figure 4(a).

2

More general than AFlow

MAS-GPT beats math-specific AFlow on MATH by 3.53% and generalizes better to other domains.

3

Low inference cost

It requires one 32B inference to build the MAS, while AFlow needs multiple proprietary LLM calls.

	SELECT	REFINE-A	REFINE-R	MATH	MMLU	GPQA
①	✗	✓	✓	60.26	77.58	36.68
②	✓	✗	✓	66.23	77.78	36.45
③	✓	✓	✗	64.90	75.96	37.15
④	✓	✓	✓	**68.65**	**78.38**	**37.62**

EXPERIMENT RESULTS

Dataset Construction Ablation

1

All three designs matter

The ablation tests inter-consistency selection, MAS adjustment Refine-A, and reasoning process Refine-R.

2

Selection has large impact

Replacing inter-consistency with random mapping causes an 8.39 absolute MATH drop.

3

Full design is best

The full row reaches the highest values shown: MATH 68.65, MMLU 78.38, and GPQA 37.62.

AvaTaR: Optimizing LLM Agents for Tool Usage via Contrastive Reasoning

Shirley Wu[§], **Shiyu Zhao**[§], **Qian Huang**[§], **Kexin Huang**[§], **Michihiro Yasunaga**[§], **Kaidi Cao**[§]
Vassilis N. Ioannidis[†], **Karthik Subbian**[†], **Jure Leskovec**[*§], **James Zou**[*§]
[*]Equal senior authorship.
[§]Department of Computer Science, Stanford University [†]Amazon

MOTIVATION

Motivation: AVATAR

1

Tool-use prompting is brittle

Complex tasks need decomposition, tool selection, and synthesis, but prompt engineering is labor-intensive.

2

Prior optimizers miss tool strategy

Existing agent optimization often does not explicitly target tool usage or multi-stage generalization.

3

Per-sample reflection is narrow

Single failed examples may not reveal systematic flaws across decomposition and tool combinations.

Core motivation AVATAR uses a comparator LLM to contrast positive and negative examples, producing holistic instructions that improve an actor's tool use.

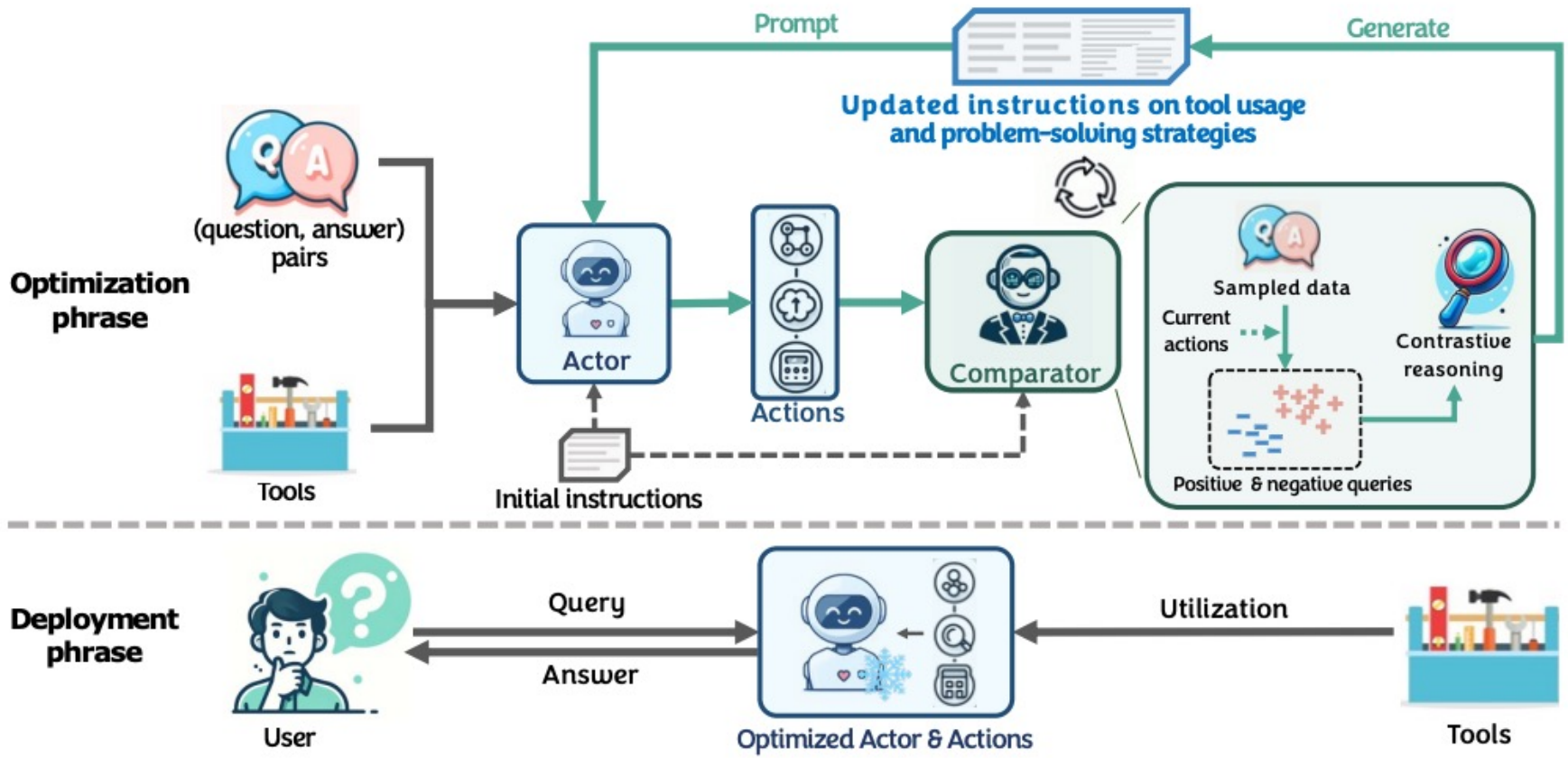

Prompt
Generate
Updated instructions on tool usage and problem-solving strategies
Optimization phrase
(question, answer) pairs
Tools
Actor
Initial instructions
Actions
Comparator
Sampled data
Current actions
Contrastive reasoning
Positive & negative queries
Deployment phrase
Query
Answer
User
Optimized Actor & Actions
Utilization
Tools

FIGURE EXPLANATION

AVATAR Overview

1. **Actor and comparator**
   The actor generates tool-using actions; the comparator analyzes successes and failures to update instructions.

2. **Contrastive optimization**
   Positive and negative queries are contrasted to identify systematic tool-use flaws.

3. **Deployment**
   The optimized actor and actions are then used directly on new queries.

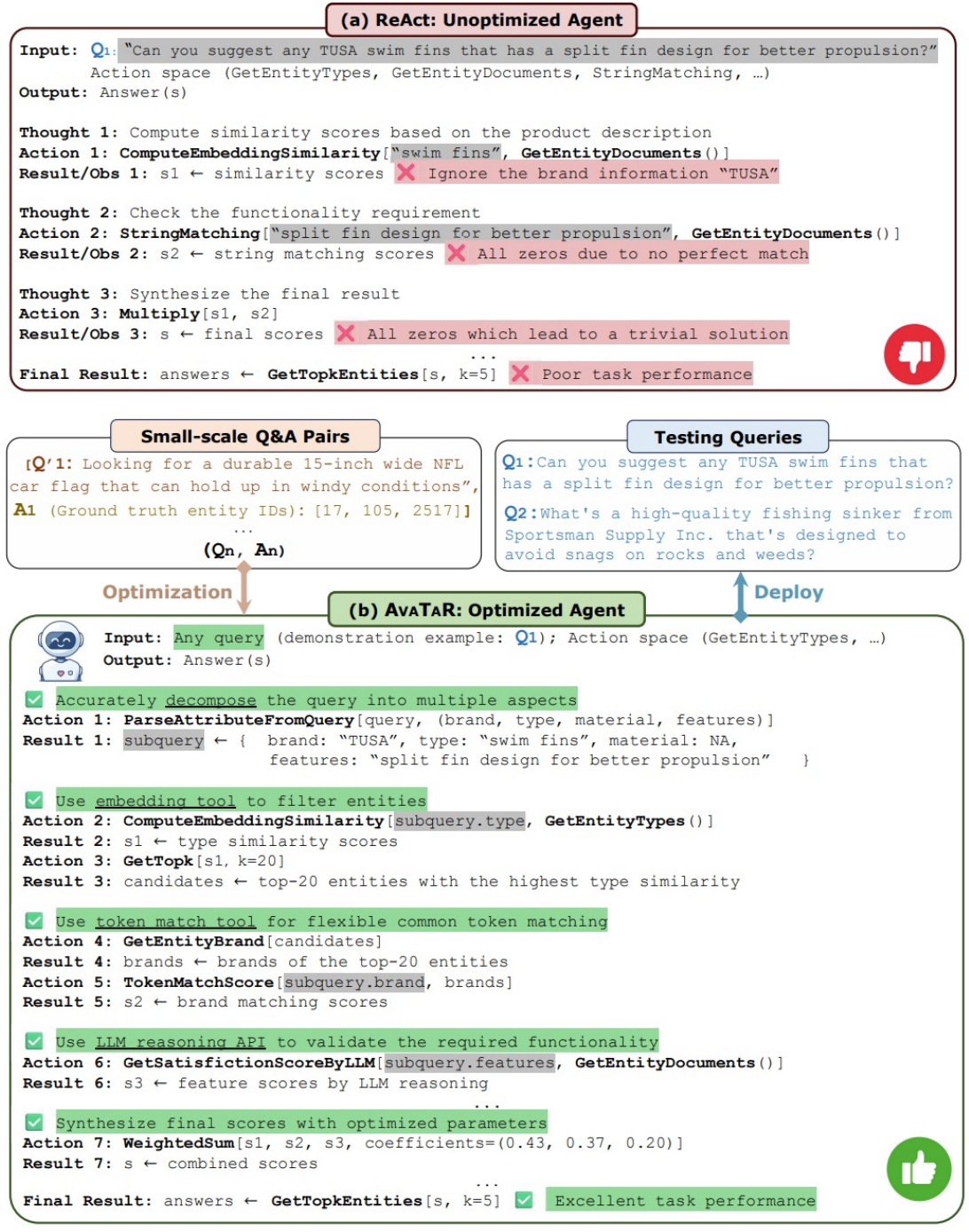

(a) ReAct: Unoptimized Agent
Input: Q1: "Can you suggest any TUSA swim fins that has a split fin design for better propulsion?"
Action space (GetEntityTypes, GetEntityDocuments, StringMatching, …)
Output: Answer(s)
Thought 1: Compute similarity scores based on the product description
Action 1: ComputeEmbeddingSimilarity["swim fins", GetEntityDocuments()]
Result/Obs 1: s1 ← similarity scores ✗ Ignore the brand information "TUSA"
Thought 2: Check the functionality requirement
Action 2: StringMatching["split fin design for better propulsion", GetEntityDocuments()]
Result/Obs 2: s2 ← string matching scores ✗ All zeros due to no perfect match
Thought 3: Synthesize the final result
Action 3: Multiply[s1, s2]
Result/Obs 3: s ← final scores ✗ All zeros which lead to a trivial solution
...
Final Result: answers ← GetTopkEntities[s, k=5] ✗ Poor task performance
Small-scale Q&A Pairs
[Q'1: Looking for a durable 15-inch wide NFL car flag that can hold up in windy conditions", A1 (Ground truth entity IDs): [17, 105, 2517]]
...
(Qn, An)
Testing Queries
Q1: Can you suggest any TUSA swim fins that has a split fin design for better propulsion?
Q2: What's a high-quality fishing sinker from Sportsman Supply Inc. that's designed to avoid snags on rocks and weeds?
Optimization
Deploy
(b) AvaTaR: Optimized Agent
Input: Any query (demonstration example: Q1); Action space (GetEntityTypes, …)
Output: Answer(s)
✓ Accurately decompose the query into multiple aspects
Action 1: ParseAttributeFromQuery[query, (brand, type, material, features)]
Result 1: subquery ← { brand: "TUSA", type: "swim fins", material: NA,
features: "split fin design for better propulsion" }
✓ Use embedding tool to filter entities
Action 2: ComputeEmbeddingSimilarity[subquery.type, GetEntityTypes()]
Result 2: s1 ← type similarity scores
Action 3: GetTopk[s1, k=20]
Result 3: candidates ← top-20 entities with the highest type similarity
✓ Use token match tool for flexible common token matching
Action 4: GetEntityBrand[candidates]
Result 4: brands ← brands of the top-20 entities
Action 5: TokenMatchScore[subquery.brand, brands]
Result 5: s2 ← brand matching scores
✓ Use LLM reasoning API to validate the required functionality
Action 6: GetSatisfictionScoreByLLM[subquery.features, GetEntityDocuments()]
Result 6: s3 ← feature scores by LLM reasoning
...
✓ Synthesize final scores with optimized parameters
Action 7: WeightedSum[s1, s2, s3, coefficients=(0.43, 0.37, 0.20)]
Result 7: s ← combined scores
...
Final Result: answers ← GetTopkEntities[s, k=5] ✓ Excellent task performance

FIGURE EXPLANATION

ReAct vs. AVATAR

1. **ReAct failure mode**
   The paper says ReAct shows incomplete decomposition and suboptimal tool combinations.

2. **Optimized action sequence**
   AVATAR decomposes the task, selects better tools, and combines scores with learned parameters.

3. **New-query transfer**
   The example illustrates how optimized actions generalize to a new user query.

	Self-Improvement	Memory	Generalization	Holistic Prompt Generation (on Tool Usage)
ReAct [55]	✗	✗	✗	✗
Self-refine [27]	✔	✗	✗	✗
Reflexion [41]	✔	✔	✗	✗
AvaTaR (Ours)	✔	✔	✔	✔

TABLE EXPLANATION

Method Capability Comparison

1

Four dimensions

The table compares self-improvement, memory, generalization, and holistic prompt generation for tool usage.

2

AVATAR covers all

AVATAR is marked as supporting all four capabilities.

3

Positioning

The comparison explains why the paper goes beyond fixed ReAct-style prompting and simple reflection.

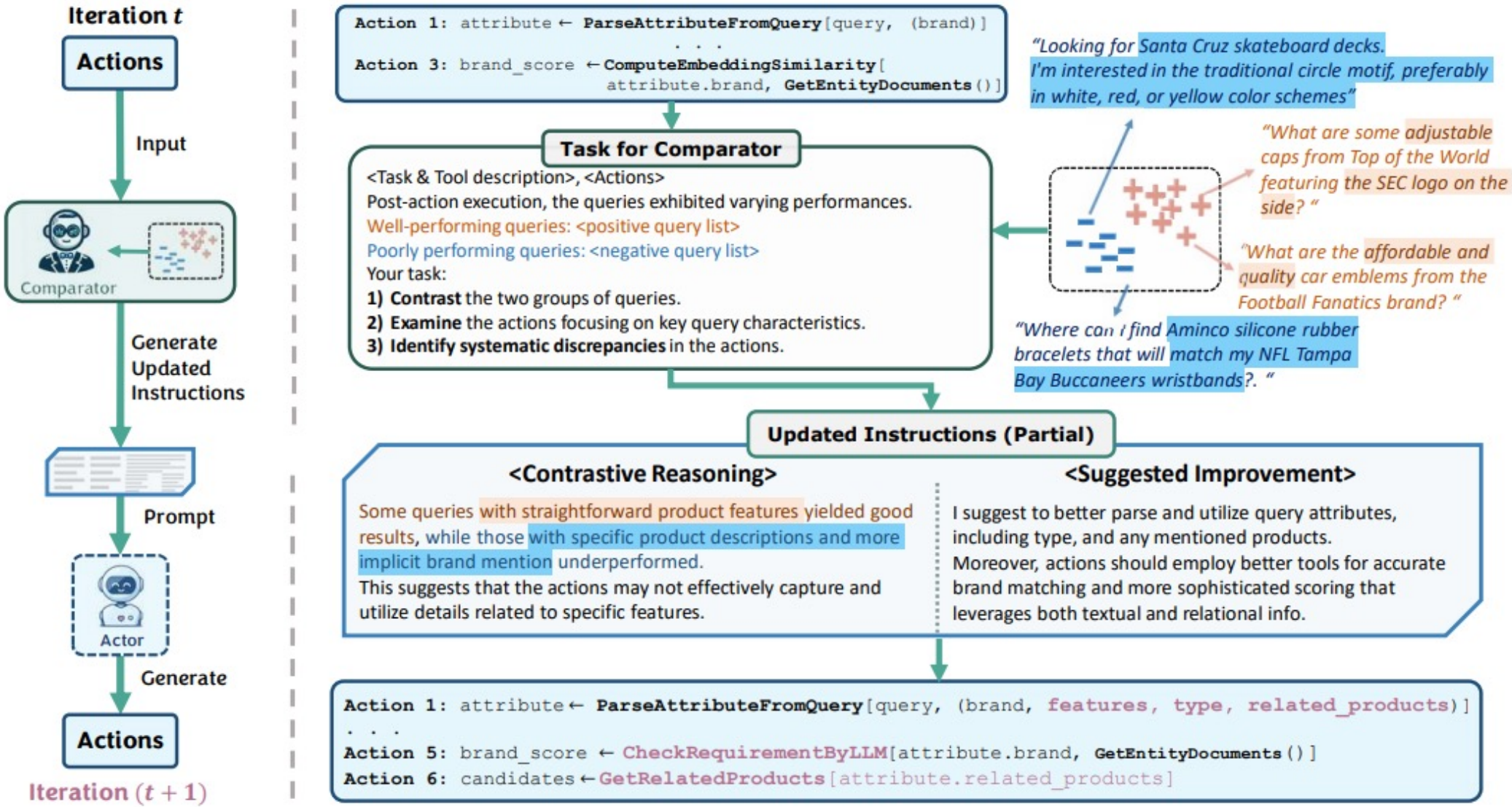

Iteration t
Actions
Input
Comparator
Generate Updated Instructions
Prompt
Actor
Generate
Actions
Iteration $(t+1)$
Action 1: attribute ← ParseAttributeFromQuery[query, (brand)]
. . .
Action 3: brand_score ← ComputeEmbeddingSimilarity[attribute.brand, GetEntityDocuments()]
Task for Comparator
<Task & Tool description>, <Actions>
Post-action execution, the queries exhibited varying performances.
Well-performing queries: <positive query list>
Poorly performing queries: <negative query list>
Your task:
1) Contrast the two groups of queries.
2) Examine the actions focusing on key query characteristics.
3) Identify systematic discrepancies in the actions.
"Looking for Santa Cruz skateboard decks. I'm interested in the traditional circle motif, preferably in white, red, or yellow color schemes"
"What are some adjustable caps from Top of the World featuring the SEC logo on the side? "
"What are the affordable and quality car emblems from the Football Fanatics brand? "
"Where can I find Aminco silicone rubber bracelets that will match my NFL Tampa Bay Buccaneers wristbands?. "
Updated Instructions (Partial)
<Contrastive Reasoning>
Some queries with straightforward product features yielded good results, while those with specific product descriptions and more implicit brand mention underperformed.
This suggests that the actions may not effectively capture and utilize details related to specific features.
<Suggested Improvement>
I suggest to better parse and utilize query attributes, including type, and any mentioned products.
Moreover, actions should employ better tools for accurate brand matching and more sophisticated scoring that leverages both textual and relational info.
Action 1: attribute ← ParseAttributeFromQuery[query, (brand, features, type, related_products)]
. . .
Action 5: brand_score ← CheckRequirementByLLM[attribute.brand, GetEntityDocuments()]
Action 6: candidates ← GetRelatedProducts[attribute.related_products]

FIGURE EXPLANATION

Comparator-Driven Instruction Updates

1. **Positive vs. negative queries**
   The comparator contrasts queries where the actor performs well and poorly.

2. **Holistic instructions**
   It generates instructions to improve decomposition, tool choice, and information incorporation.

3. **Tool-use correction**
   The paper's example replaces an imprecise embedding match with LLM verification for better brand matching.

Experiments

	AMAZON				MAG				PRIME			
	Hit@1	**Hit@5**	**R@20**	**MRR**	**Hit@1**	**Hit@5**	**R@20**	**MRR**	**Hit@1**	**Hit@5**	**R@20**	**MRR**
DPR	15.29	47.93	44.49	30.20	10.51	35.23	42.11	21.34	4.46	21.85	30.13	12.38
QAGNN	26.56	50.01	52.05	37.75	12.88	39.01	46.97	29.12	8.85	21.35	29.63	14.73
ada-002	39.16	62.73	53.29	50.35	29.08	49.61	48.36	38.62	12.63	31.49	36.00	21.41
multi-ada-002	40.07	64.98	55.12	51.55	25.92	50.43	**50.80**	36.94	15.10	33.56	38.05	23.49
ReAct	42.14	64.56	50.81	52.30	31.07	49.49	47.03	39.25	15.28	31.95	33.63	22.76
Reflexion	42.79	65.05	54.70	52.91	40.71	54.44	49.55	47.06	14.28	34.99	38.52	24.82
AVATAR-C	40.92	63.63	53.68	51.73	33.25	52.17	47.88	41.34	8.82	23.82	30.32	16.20
AVATAR	**49.87**	**69.16**	**60.57**	**58.70**	**44.36**	**59.66**	50.63	**51.15**	**18.44**	**36.73**	**39.31**	**26.73**
Relative Improvement	16.6%	6.3%	9.9%	12.2%	9.6%	2.1%	-0.3%	8.7%	20.7%	5.0%	2.1%	7.7%

EXPERIMENT RESULTS

STARK Retrieval Results

1

State-of-the-art retrieval

On Amazon, MAG, and Prime, AVATAR outperforms leading retrieval and agent baselines.

2

Reported average gains

The paper reports average improvements of 15.6% on Hit@1 and 9.5% on MRR.

3

Comparator matters

The ablation variant AVATAR-C is lower, showing the importance of comparator-generated instructions.

	Hit@1	Hit@5	R@20	MRR
`clip-vit-large-patch14`	37.2	<u>56.4</u>	72.8	<u>46.3</u>
ReAct (`claude3`)	<u>38.8</u>	54.8	71.6	46.1
Reflexion (`claude3`)	28.4	53.2	75.2	41.2
AvaTaR-C (`claude3`)	28.8	53.2	<u>78.4</u>	40.0
AvaTaR (`claude3`)	**42.4**	**63.0**	**79.2**	**52.3**
Relative Improvement	9.2%	11.7%	5.3%	13.0%

EXPERIMENT RESULTS

Flickr30K Entities

1

Best shown metrics

AVATAR reaches 42.4 Hit@1, 63.0 Hit@5, 79.2 Recall@20, and 52.3 MRR.

2

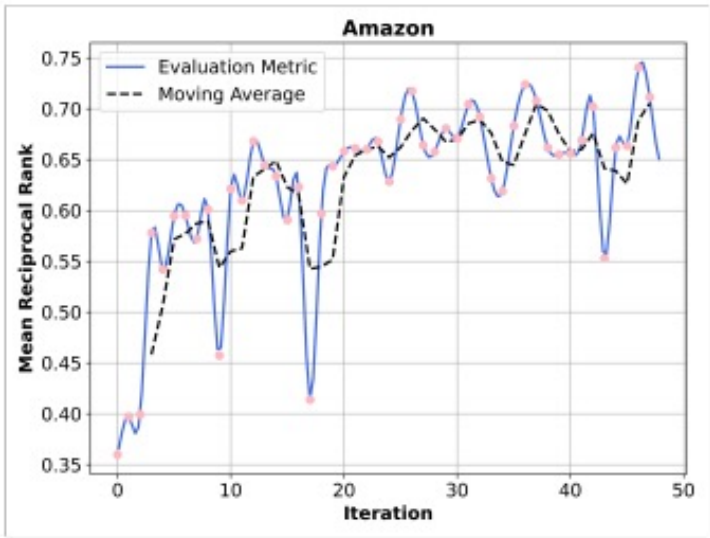
Relative gains

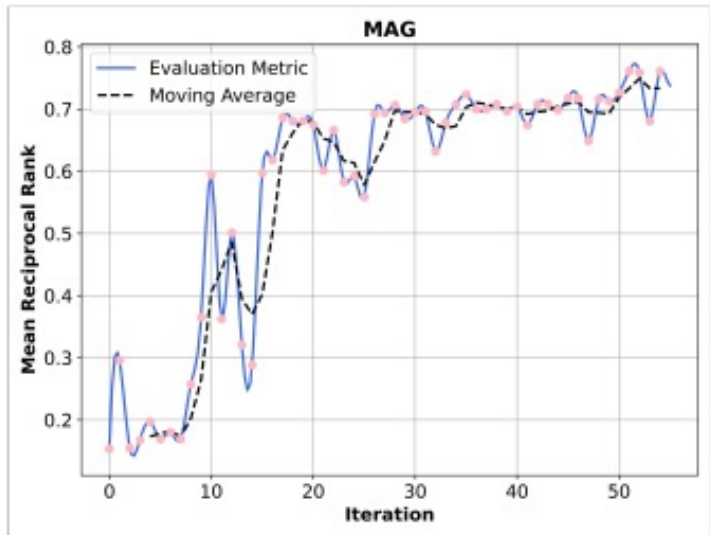
The row reports gains of 9.2%, 11.7%, 5.3%, and 13.0% over the best prior metric values.

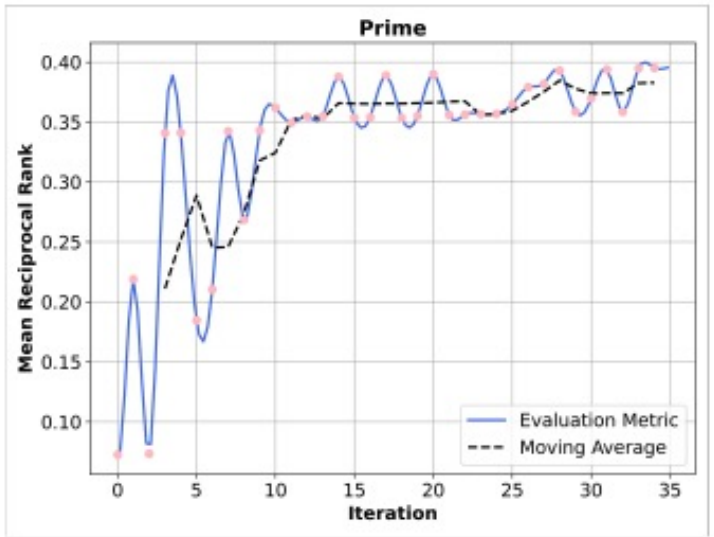
3

Why batch contrast helps

The paper says Reflexion can overfit to specific images, while AVATAR uses batch-wise contrastive reasoning.


Amazon
Evaluation Metric
Moving Average
Mean Reciprocal Rank
Iteration
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0
10
20
30
40
50


MAG
Evaluation Metric
Moving Average
Mean Reciprocal Rank
Iteration
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
10
20
30
40
50


Prime
Evaluation Metric
Moving Average
Mean Reciprocal Rank
Iteration
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0
5
10
15
20
25
30
35

EXPERIMENT RESULTS

Optimization Dynamics

1

Agents improve during optimization

The paper reports validation improvements such as 35% to 75% on Amazon and 20% to 78% on MAG.

2

Memory encourages convergence

A memory bank stores past best-performing actions and helps the actor converge by the end of optimization.

3

High-level tool behavior

On Flickr30K, AVATAR develops actions such as IDF-based reweighting for phrase matching.

	HotpotQA	ArxivQA	ToolQA			
			SciREX-easy	SciREX-hard	Agenda-easy	Agenda-hard
CoT	28.0%	58.0%	1.7%	0.0%	0.0%	0.0%
ReAct	40.0%	72.0%	31.7%	17.5%	38.3%	3.33%
Reflexion	46.0%	77.0%	28.3%	13.3%	30.0%	3.33%
ExpeL	39.0%	73.0%	36.7%	14.5%	56.6%	1.67%
Retroformer (#retry=1)	51.0%	-	-	-	-	-
AvaTaR-C	41.0%	73.0%	31.7%	13.3%	31.7%	1.67%
AvaTaR	**53.0%**	**84.0%**	**37.5%**	**23.3%**	**60.0%**	**4.17%**
Relative Improvement	3.92%	9.09%	2.18%	33.1%	5.82%	25.0%

EXPERIMENT RESULTS

Question Answering Results

1

Consistent QA gains

AVATAR outperforms baselines on HotpotQA, ArxivQA, and ToolQA variants.

2

Largest ToolQA gains

The paper highlights 33.1% relative gain on SciREX-Hard and 25.0% on Agenda-Hard.

3

Interpretation

The gains are attributed to optimized prompts that improve context understanding across question types.

Improved Divide and Conquer (4/25)

Actions perform well on **queries that more often mention noun concepts like 'woman', 'man'.** The queries with poorer results often **cover broader set of other concepts, e.g., actions (ride, cook), and attributes (blue, light).**

Consider separation extraction on adjectives / compound nouns / verb phrases to extract more informative concepts for negative queries.

Added actions (summary)
1) **Identify the following query patterns separately**
   NP (Basic Noun Phrase) : {<DT|PP\$>?<JJ>*<NN.*>+}
   CP (Comparative Phrase) : {<JJ>?<NN.*>+<IN><JJ>?}
   VP (Verb Phrase) : {<VB.*><NP|PP>+}
2) Correspond them with the image attributes precisely

Sensible Tool Differentiation (16/25)

Action 1: **VqaByLLM**(GetImages(), "What is the location?")
Result 1: location ← Long description about the location
Action 2: **StringMatch**(location, query_attribute.location)

Actions performs well on **queries without requesting images from specific locations,** while fail on **queries that ask for locations such as "square", "street", "stadium".**

This suggests issues with extracting image locations, possibly due to VqaByLLM's lengthy outputs and StringMatch's overly strict criteria.

Action 1: GetVisualAttributesByLLM(GetImages(), "location")
Result 1: location ← Concise location attribute
Action 2: **TokenMatchScore** (location, query_attribute.location)

Numerical Parameter Learning (5/25)

parameters = {phrase_weight: 0.5, visual_weight: 0.5}
WeightedSum(phrase_scores, visual_scores, parameters)

Actions performs well on **queries mentioning objects, such as animals and people (e.g. dog, man, woman)** But they struggles with **more abstract or scenery focused queries (e.g. people relaxing on grass, person jumping near a car).**

Emphasize on the visual attributes by increasing the **visual_weight**. This will help capture the overall scene elements better.

parameters = {phrase_weight: **0.4**, visual_weight: **0.6**}
WeightedSum(phrase_scores, visual_scores, parameters)

FIGURE EXPLANATION

Comparator Instruction Types

1

Three recurring categories

The paper groups comparator instructions into divide-and-conquer, tool differentiation, and parameter learning.

2

Measured over iterations

Occurrences are counted across 25 optimization iterations on Flickr30K Entities.

3

Tool-use focus

The examples show comparator instructions improving retrieval strategy, tool selection, and score synthesis.

	AMAZON				MAG				PRIME			
	Hit@1	**Hit@5**	**R@20**	**MRR**	**Hit@1**	**Hit@5**	**R@20**	**MRR**	**Hit@1**	**Hit@5**	**R@20**	**MRR**
DPR (roberta)	16.05	39.51	15.23	27.21	4.72	9.52	25.00	7.90	2.04	9.18	10.69	7.05
ada-002	39.50	64.19	35.46	52.65	28.57	41.67	35.95	35.81	17.35	34.69	41.09	26.35
multi-ada-002	46.91	72.84	40.22	58.74	23.81	41.67	**39.85**	31.43	24.49	39.80	47.21	32.98
ReAct	45.65	71.73	35.95	58.81	27.27	40.00	35.95	33.94	21.73	33.33	41.09	28.20
Reflexion	49.38	64.19	35.95	58.96	28.57	39.29	35.95	36.53	16.52	33.03	41.09	23.99
AVATAR	**58.32**	**76.54**	**42.43**	**65.91**	**33.33**	**42.86**	35.94	**38.62**	**33.03**	**51.37**	**53.34**	**41.00**
Rel. Impr.	17.5%	5.1%		11.8%	16.7%	2.9%		5.7%	28.7%	27.3%		21.4%

EXPERIMENT RESULTS

Leave-Out Query Generalization

1

Human-generated leave-out queries

The appendix evaluates optimized actions on STARK queries that differ from the training queries.

2

Reported generalization gain

The paper reports an average 20.9% Hit@1 improvement on leave-out queries.

3

Conclusion

Comparator instructions are designed at the group level, which helps transfer to novel query patterns.