ICLR 2026 Poster

Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models

Qizheng Zhang, Changran Hu, et al.

Stanford University | SambaNova Systems, Inc. | UC Berkeley

Presented by Yejin Yoon

Pre-Requisite

Two Ways to Improve an LLM

What Makes a "Good" Context?

Pre-requisites — Two Ways to Improve an LLM

How do we make an LLM smarter? — Two paradigms for improving LLMs

Both aim to make an LLM perform better on a target task, but in fundamentally different ways

🔥 Weight Adaptation

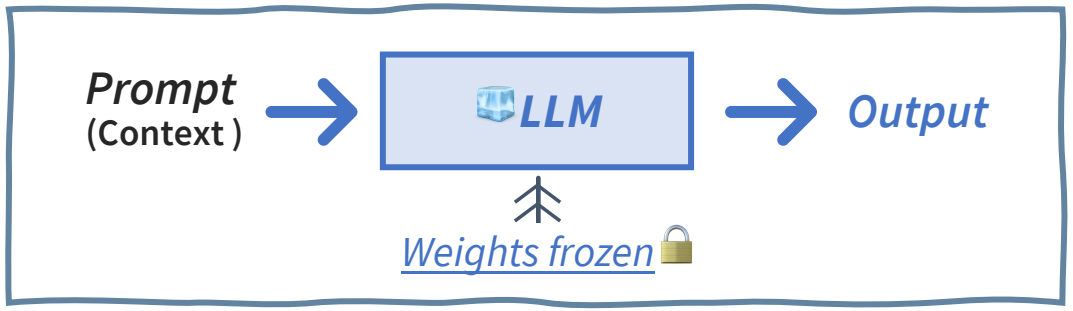
"Modify the model itself"

- Weights are updated (changed) 🆕
- Hard to undo, audit
- Expensive (GPU-heavy) 💰

→ e.g. Fine-tuning, LoRA, RLHF, …

📝 Context Adaptation

"Modify what the model sees"



- Weights are frozen 🥶
- Interpretable, easy to revise
- Cheap (just inference) 👍

→ e.g. ICL, CoT, RAG, Agent memory, …

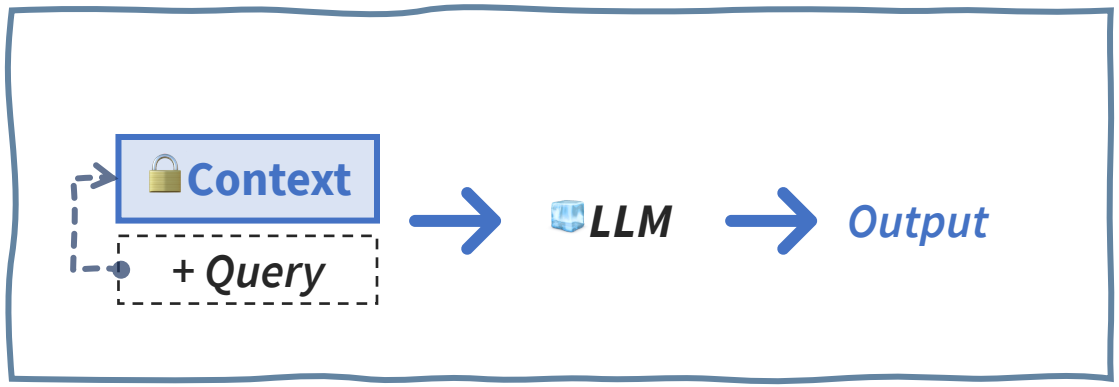
Two paradigms, one principle: change the ***model***, or change its ***input***.

Pre-requisites — What Makes a 'Good' Context?

Not all contexts are created equal🤔

Same goal — two ways to provide context to an LLM.

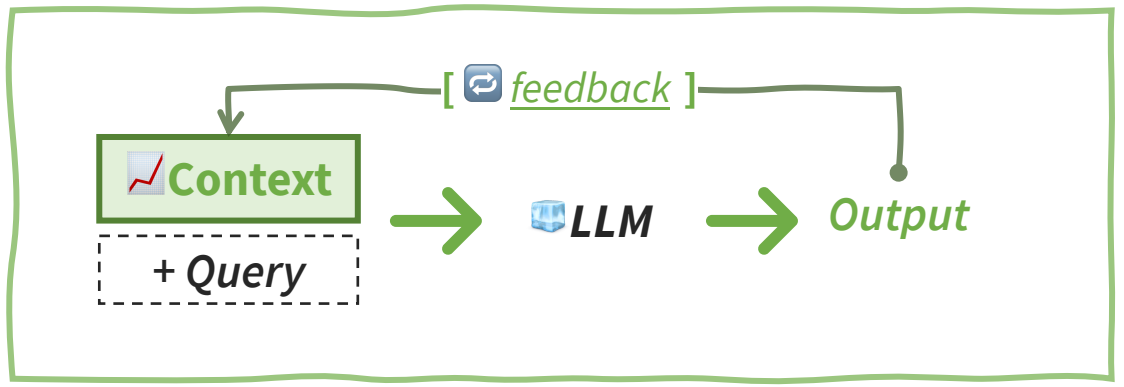
Static Context – *fixed prompt*



- Context is fixed; *only the query varies*
- No update from experience

(e.g. system prompt, few-shot examples)

Evolving Context – *updates over time*



- Learns from each execution; *Auto-updated*
- Self-improving

(e.g. agent memory, accumulated playbook)

A good context should grow with experience. But how — automatically?

Contents

Background

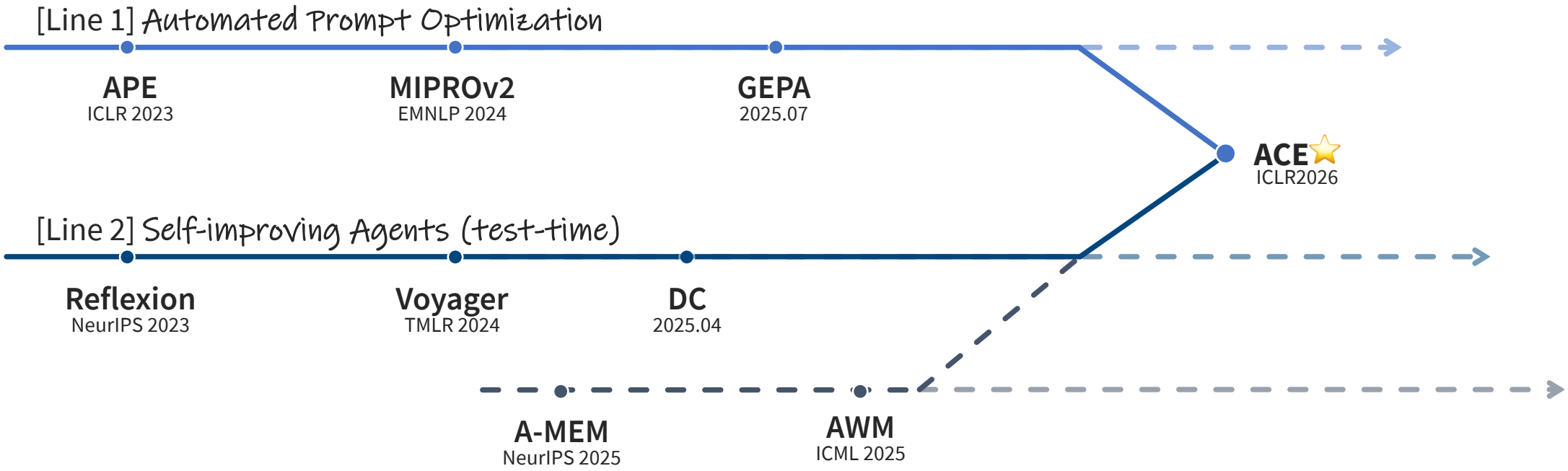
Where does ACE sit in the literature?

Two converging research lines

Background

The Real Track — Self-Improving LLM Agents

The authors call it "***context adaptation***" — but the real landscape is two converging lines:

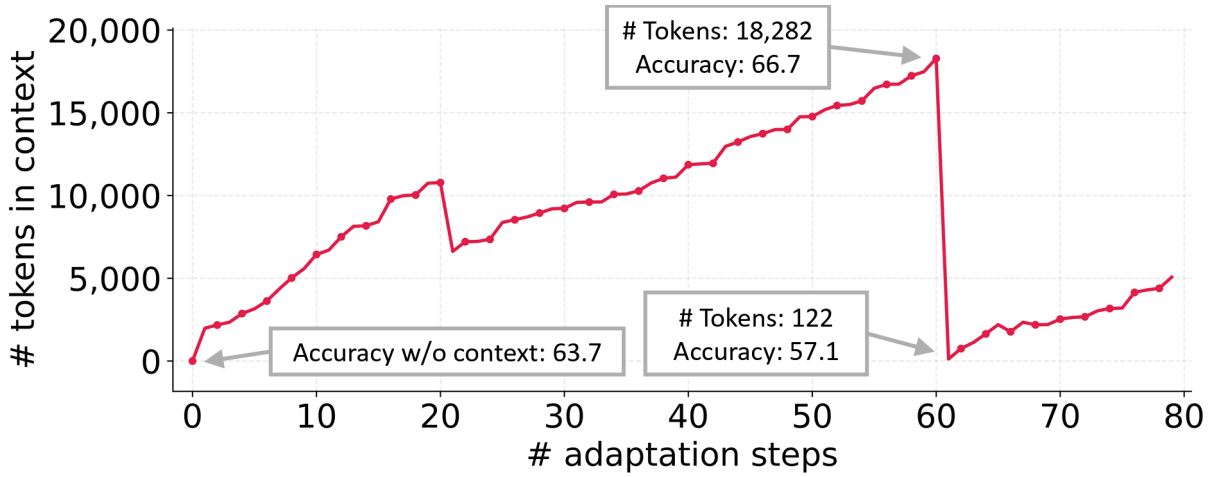


We follow Line 2 — ACE's core idea (*incremental memory*) extends Line 2's lineage.

Background

[Line 2] The Self-Improving Agents Track : 2 failure modes of existing methods

1. **Reflexion** (2023.03) : per-task verbal self-reflection → ***no cross-task accumulation***
2. **Dynamic Cheatsheet** (2025.04) : accumulating cross-task memory → ***but collapses on rewrite***
3. **ACE** (2025.10) : structured, incremental playbook → *heavy reflector dependence*
4. **CLEAR** (2026.04) : trains a separate context model ; a different turn



✓ **Brevity Bias** (Gao et al. 2025)

: Iterative optimization collapses to *short generic prompts*

e.g. *"Create unit tests to ensure methods behave as expected"*

✓ **Context Collapse** (◀ Fig. 2)

: LLM rewrites entire context
→ *suddenly* compresses tokens *too much*

e.g. [step 60] 18,282 tokens → 66.7%
[step 61] 122 tokens → 57.1% (worse than baseline!)

Each step solved the previous one's failure; **ACE** solves DC's biggest one — *collapse*.

Method — ACE Framework

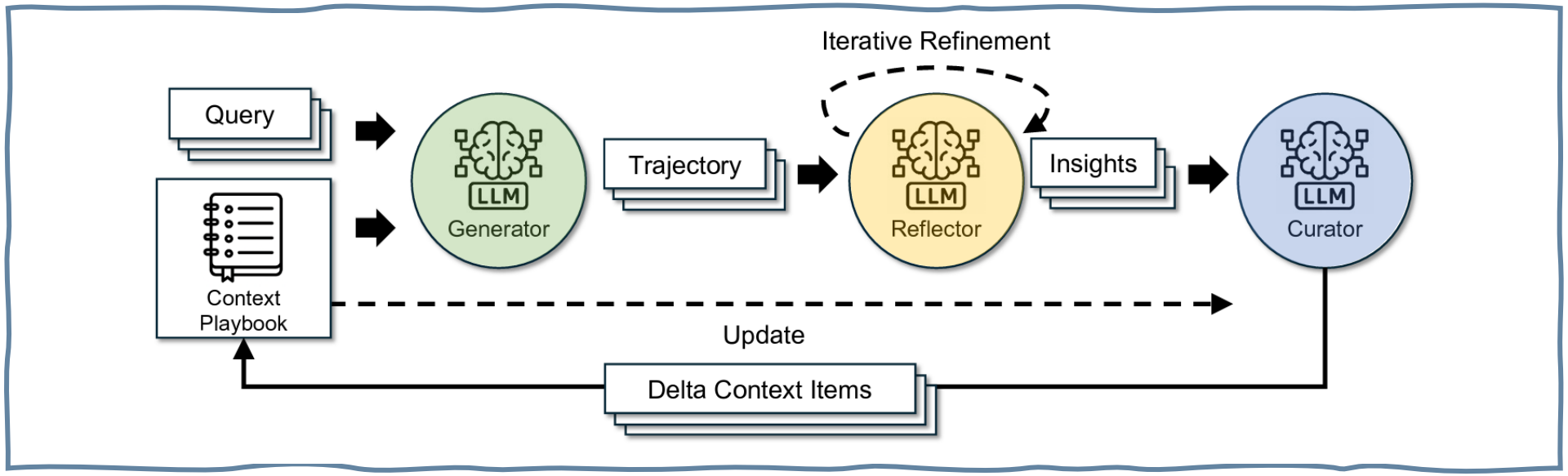
Three-role architecture

Incremental delta updates

Grow-and-refine

ACE Overview — "Context as a Living **Playbook**"

Try → Reflect → Curate — and update the playbook *incrementally*



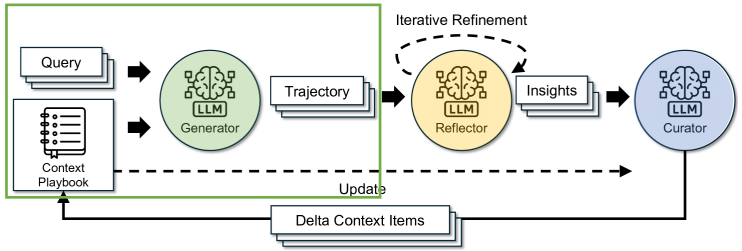
1. **Generator** : Produces a reasoning trajectory — what worked, what misled
2. **Reflector** : Diagnoses successes and failures into reusable insights
3. **Curator** : Adds insights as bullets — without rewriting the *whole* context

Same LLM, different prompts — accumulate, organize, and *incrementally* update.

ACE Framework

[Generator] produce reasoning trajectory for new query → Reflector → Curator

- **Input** : query + current playbook
- **Output** : trajectory (which bullets helped, which misled)



Task Query "Count all playlists in Spotify" ✅ Ans.: 247

Current Playbook [Basic auth + API calling guidelines]

Generator's attempt (Trajectory)

```
playlists = []
for i in range(10):                    ← Generator wrote this
        page = api.get_playlists(page=i)
        playlists.extend(page.items)
return len(playlists)
```

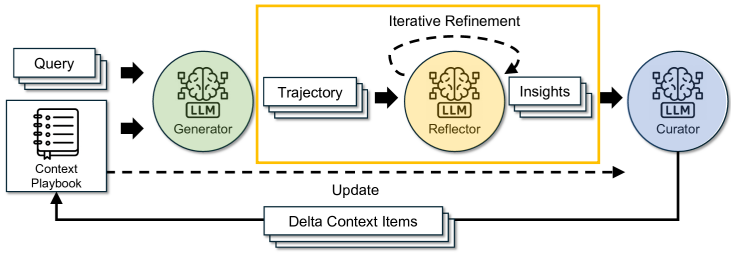
Result 100 ✗

By trying, the **Generator** exposes what the playbook is *missing*.

ACE Framework

Generator → [**Reflector**] extract concrete insights from successes/failures → Curator

- **Input** : trajectory (from ***Generator***) + ***Reflector*** 's own previous insights (iterative)
- **Output** : concrete, reusable insights



Trajectory — From Generator **Code** + **result (100)**

Diagnosis — ✗ **Used range (10) instead of pagination**

Root Cause — **Agent assumed playlists fit in 10 pages**

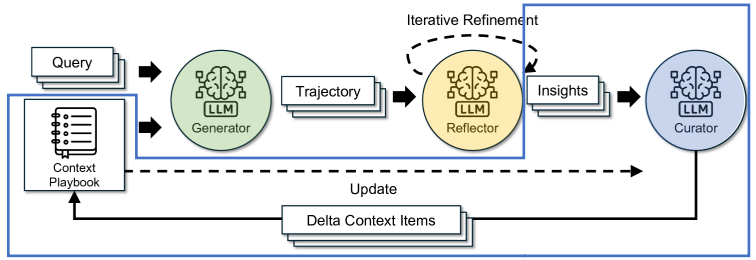
Key Insights — Many APIs return paginated results.
Use while-loop with *has_next_page* check,
Not a hard-coded range.

By diagnosing, the **Reflector** turns failures into *reusable* lessons.

ACE Framework

Generator → Reflector → [Curator] Incremental Delta Updates

- **Input** : insights (from ***Reflector***) + current playbook
- **Output** : updated playbook (delta items merged in)



bullets = (*metadata*: id, counters) + (*content*)

- **Localization**: only relevant bullets updated
- **Fine-grained retrieval**: focus on pertinent knowledge
- **Incremental**: efficient merge/prune/deduplicate

Deterministic merge logic (non-LLM!)

Insight

Many APIs return paginated results. …

Curator Decides

- This is a NEW insight (not in current playbook)
- Add to "apis_to_use_for_specific_info" section

New bullet

[api-00027] Pagination handling
About pagination: many APIs return items in "pages".
Use while True loop until no more results,
not for i in range(N).
helpful=0, harmful=0 (counters initialized)

**** Other bullets ****

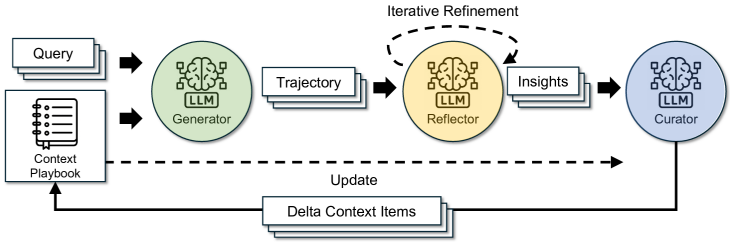
⭐ ***Not touched*** **(δ update – no rewrite)**

No rewrite, no collapse — just a small bullet appended.

ACE Framework

Generator → Reflector → Curator → Grow-and-Refine

- **Grow** : new ID → append new bullet
- **Refine** : existing bullet → in-place update (helpful counter ++)
  semantic duplicates → merged via embedding-based deduplication
- **Mode** : *proactive* (every step) vs. *lazy* (when context is full)



- Later, **Generator** faces another pagination task :
  → Uses bullet [api-00027] correctly → helpful counter += 1
- Existing bullet about "*page_index* loop" :
  → Semantically duplicate with [api-00027]
  → **Merged** via embedding-based deduplication

The playbook grows when needed, refines when redundant.

ACE Framework

ACE Playbook — Real Example

The Generated ACE Playbook on AppWorld

STRATEGIES AND HARD RULES

[shr-00009]

When processing time-sensitive transactions involving specific relationships: always resolve identities from the correct source app (phone contacts), use proper datetime range comparisons instead of string matching, and verify all filtering criteria (relationship + time) are met before processing items. This ensures accurate identification and processing of the right transactions.

USEFUL CODE SNIPPETS AND TEMPLATES

[code-00013]

For efficient artist aggregation when processing songs, use defaultdict(list) to map song titles to artist names:

```
from collections import defaultdict artist_map = defaultdict(list) for song in songs: artist_map[song['title']].extend([artist['name'] for artist in song['artists']])
```

TROUBLESHOOTING AND PITFALLS

[ts-00003]

If authentication fails, troubleshoot systematically: try phone number instead of email as username, clean credentials from supervisor, check API documentation for correct parameters etc. Do not proceed with workarounds.

Auto-generated by the system from execution feedback.

Experimental Results

Agent benchmark

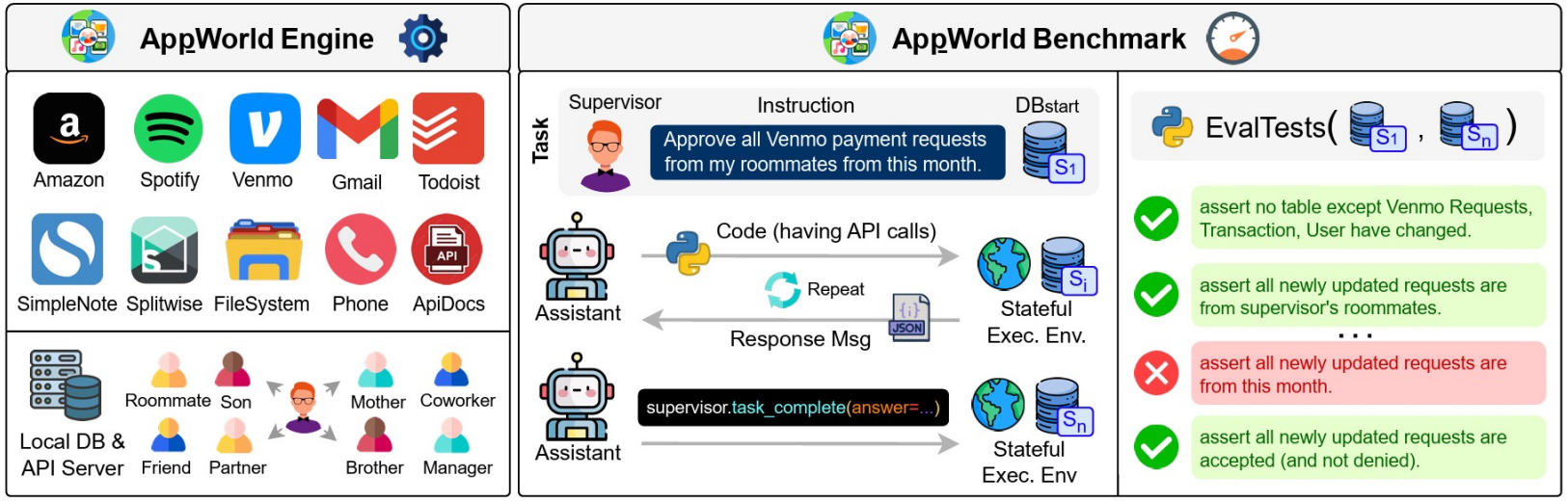
Domain-specific reasoning

Cost & efficiency

Experimental Setup

Benchmarks · Baselines

- **Benchmarks :** [**Agent**] **AppWorld** (TGC, SGC; normal & test-challenge)
  [**Domain**] **FiNER** (XBRL labeling), **Formula** (numerical reasoning)
- **Baselines :** Base, ICL, MIPROv2, GEPA(key offline baseline), DC (key online baseline)
- **Backbones :** DeepSeek-V3.1 (same model for Generator / Reflector / Curator)
- **Evaluation :** Offline — playbook w/training set, Online — execution feedback only



✓ **TGC** (Task Goal Completion)
→ Did the agent achieve the task?

✓ **SGC** (Scenario Goal Completion)
→ Did all sub-tasks succeed?

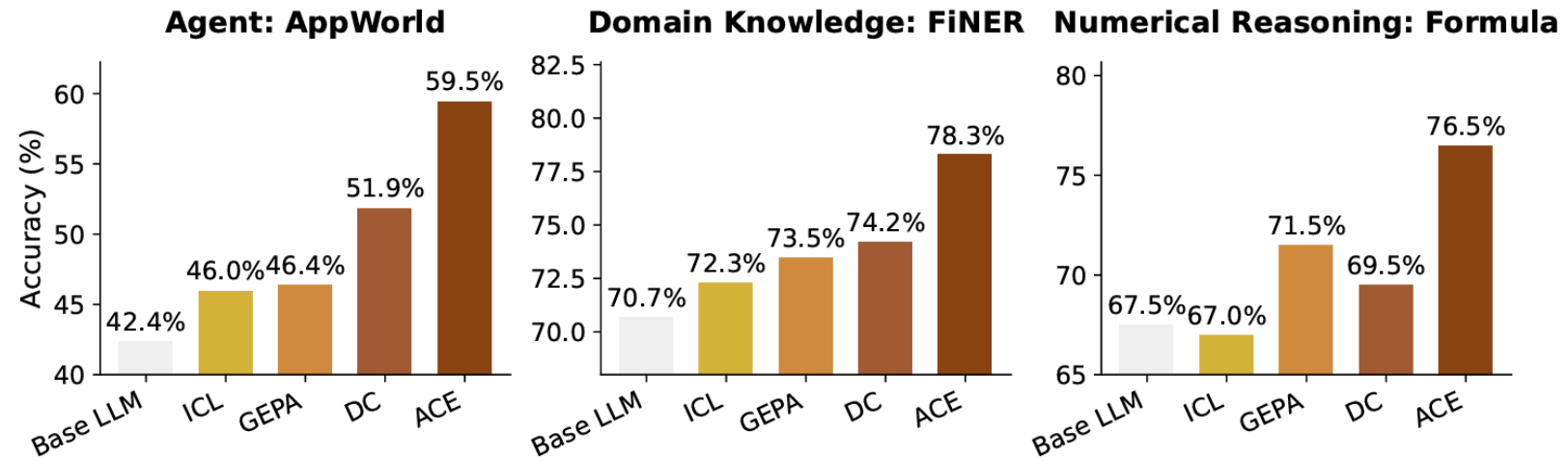
e.g. **< Task >**
*"Help me follow my **Spotify** friends on **Venmo** if they have an account."*

→ Agent must: (1) query ***Spotify*** API
(2) cross-check ***Venmo*** accounts
(3) send follow requests

Experimental Setup

Benchmarks · Baselines

- **Benchmarks** : **[Agent] AppWorld** (TGC, SGC; normal & test-challenge)
  [Domain] FiNER (XBRL labeling), **Formula** (numerical reasoning), **DDXPlus** (medical), **BIRD** (SQL)
- **Baselines** : Base, ICL, MIPROv2, GEPA(key offline baseline), DC (key online baseline)
- **Backbones** : DeepSeek-V3.1 (same model for Generator / Reflector / Curator)
- **Evaluation** : Offline — playbook w/training set, Online — execution feedback only



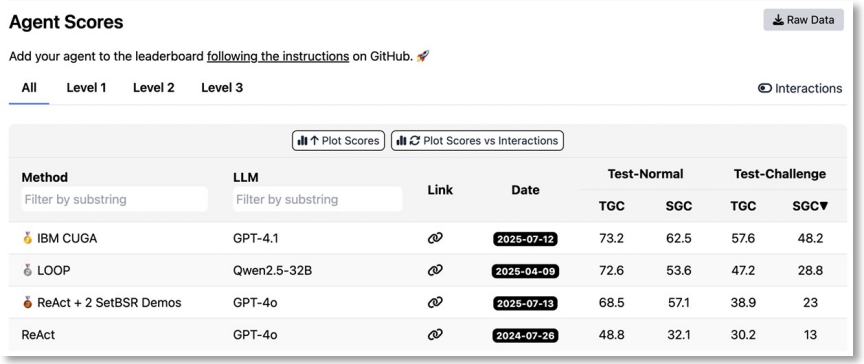
Effects

Agent Benchmark : AppWorld Results

Method	GT Labels	Test-Normal		Test-Challenge		Average
		TGC↑	SGC↑	TGC↑	SGC↑	
DeepSeek-V3.1-671B as Base LLM						
ReAct		63.7	42.9	41.5	21.6	42.4
Offline Adaptation						
ReAct + ICL	✓	64.3$_{+0.6}$	46.4$_{+3.5}$	46.0$_{+4.5}$	27.3$_{+5.7}$	46.0$_{+3.6}$
ReAct + GEPA	✓	64.9$_{+1.2}$	44.6$_{+1.7}$	46.0$_{+4.5}$	30.2$_{+8.6}$	46.4$_{+4.0}$
ReAct + ACE	✓	**76.2**$_{+12.5}$	**64.3**$_{+21.4}$	**57.3**$_{+15.8}$	**39.6**$_{+18.0}$	**59.4**$_{+17.0}$
ReAct + ACE	✗	75.0$_{+11.3}$	**64.3**$_{+21.4}$	54.4$_{+12.9}$	35.2$_{+13.6}$	57.2$_{+14.8}$
Online Adaptation						
ReAct + DC (CU)	✗	65.5$_{+1.8}$	**58.9**$_{+16.0}$	52.3$_{+10.8}$	30.8$_{+9.2}$	51.9$_{+9.5}$
ReAct + ACE	✗	**69.6**$_{+5.9}$	53.6$_{+10.7}$	**66.0**$_{+24.5}$	**48.9**$_{+27.3}$	**59.5**$_{+17.1}$

Table 1: **Results on the AppWorld Agent Benchmark (DeepSeek-V3.1-671B as the Base LLM).** "GT labels" indicates whether ground-truth labels are available to the Reflector during adaptation. We evaluate the ACE framework against multiple baselines on top of the official ReAct implementation, both for offline and online context adaptation. ReAct + ACE outperforms selected baselines by an average of 10.6%, and could achieve good performance even without access to GT labels.

Online no-labels 59.5
≈ *Offline with-labels* 59.4 ✅



✓ Leaderboard(2025/09)

IBM CUGA (GPT-4.1) 60.3 ≈ *ACE* (DeepSeek-V3.1) 59.4 ✅

Test-challenge : *ACE* (DeepSeek-V3.1) +8.4pp 🔺

Better context 〉 bigger model → No supervision needed — execution feedback is enough

Effects

Domain Benchmark : FiNER, Formula, DDXPlus, BIRD-SQL Results

Method	GT Labels	FiNER (Acc↑)	Formula (Acc↑)	Average
DeepSeek-V3.1 as Base LLM				
Base LLM		70.7	67.5	69.1
Offline Adaptation				
ICL	✓	72.3$_{+1.6}$	67.0$_{-0.5}$	69.6$_{+0.5}$
MIPROv2	✓	72.4$_{+1.7}$	69.5$_{+2.0}$	70.9$_{+1.8}$
GEPA	✓	73.5$_{+2.8}$	71.5$_{+4.0}$	72.5$_{+3.4}$
ACE	✓	**78.3**$_{+7.6}$	**85.5**$_{+18.0}$	**81.9**$_{+12.8}$
ACE	✗	71.1$_{+0.4}$	83.0$_{+15.5}$	77.1$_{+8.0}$
Online Adaptation				
DC (CU)	✓	74.2$_{+3.5}$	69.5$_{+2.0}$	71.8$_{+2.7}$
DC (CU)	✗	68.3$_{-2.4}$	62.5$_{-5.0}$	65.4$_{-3.7}$
ACE	✓	**76.7**$_{+6.0}$	76.5$_{+9.0}$	**76.6**$_{+7.5}$
ACE	✗	67.3$_{-3.4}$	**78.5**$_{+11.0}$	72.9$_{+3.8}$

Table 2: **Results on Financial Analysis Benchmark (DeepSeek-V3.1-671B as the Base LLM).** "GT labels" indicates whether ground-truth labels are available to the Reflector during adaptation.

Method	Accuracy (↑)
DeepSeek-V3.1 as Base LLM	
Base LLM	75.2
Offline Adaptation	
GEPA	76.4$_{+1.2}$
ACE	**90.2**$_{+15.0}$

Table 10: **Results on Medical Reasoning Benchmark (DeepSeek-V3.1-671B as the Base LLM).** We use DDXPlus from StreamBench.

Method	Simple (↑)	Moderate (↑)	Challenging (↑)	Average
DeepSeek-V3.1 as Base LLM				
Base LLM	46.4	48.2	55.1	47.8
Offline Adaptation				
GEPA	51.6$_{+5.2}$	**51.9**$_{+3.7}$	**57.2**$_{+2.1}$	52.2$_{+4.4}$
ACE	**53.5**$_{+7.1}$	50.7$_{+2.5}$	56.6$_{+1.5}$	**52.9**$_{+5.1}$

Table 11: **Results on Text-to-SQL Benchmark (DeepSeek-V3.1-671B as the Base LLM).** We use BIRD-SQL from StreamBench.

ACE 〉〉 all baselines on Finance; Generalizes beyond finance — same gains hold

Effects

Ablation — What's Doing the Work?

Method	GT Labels	Test-Normal		Test-Challenge		Average
		TGC↑	SGC↑	TGC↑	SGC↑	
DeepSeek-V3.1 as Base LLM						
ReAct		63.7	42.9	41.5	21.6	42.4
Offline Adaptation						
ReAct + ACE w/o Reflector or multi-epoch	✓	$70.8_{+7.1}$	$55.4_{+12.5}$	$55.9_{+14.4}$	$38.1_{+17.5}$	$55.1_{+12.7}$
ReAct + ACE w/o multi-epoch	✓	$72.0_{+8.3}$	$60.7_{+17.8}$	$54.9_{+13.4}$	$39.6_{+18.0}$	$56.8_{+14.4}$
ReAct + ACE	✓	$76.2_{+12.5}$	$64.3_{+21.4}$	$57.3_{+15.8}$	$39.6_{+18.0}$	$59.4_{+17.0}$
Online Adaptation						
ReAct + ACE	✗	$67.9_{+4.2}$	$51.8_{+8.9}$	$61.4_{+19.9}$	$43.2_{+21.6}$	$56.1_{+13.7}$
ReAct + ACE + offline warmup	✗	$69.6_{+5.9}$	$53.6_{+10.7}$	$66.0_{+24.5}$	$48.9_{+27.3}$	$59.5_{+17.1}$

Table 3: **Ablation Studies on AppWorld.** We study how particular design choices of ACE (iterative refinement, multi-epoch adaptation, and offline warmup) could help high-quality context adaptation.

- Without Reflector + multi-epoch: 55.1 (still +12.7) vs. Full **ACE**: 59.4 (+17.0)
- Critical: w/o incremental updates → drop -27.8% ***SGC*** (Appendix A.5)

Incremental delta updates are the core innovation

Effects

Cost and Speed

Method	Latency (s)↓	# Rollouts↓
ReAct + GEPA	53898	1434
ReAct + ACE	9517 (-82.3%)	357 (-75.1%)

(a) **Offline** (AppWorld).

Method	Latency (s)↓	Token Cost ($)↓
DC (CU)	65104	17.7
ACE	5503 (-91.5%)	2.9 (-83.6%)

(b) **Online** (FiNER).

Table 4: **Cost and Speed Analysis.** We measure the context adaptation latency, number of rollouts, and dollar costs of ACE against GEPA (offline) and DC (online).

- Adaptation
  - vs. GEPA: -82.3% latency, -75.1% rollouts, -80.8% input tokens
  - vs. DC (online): -91.5% latency, -83.6% cost
- KV cache* reuse: 91.8% input tokens cached → -82.6% billed cost

Curator only appends new bullets
→ existing playbook prefix stays the same
→ 91.8% of input tokens hit the KV cache
→ -82.6% billed cost (cached tokens cost much less)

* KV cache : LLM serving infra reuses computation for unchanged prefix tokens

More tokens ≠ more cost in modern serving infra

Effects

Robustness Checks

LLM Generalization	Reflector Quality	Hyperparameters
4 LLM families tested	↙ Weaker reflector still helps (+5.9 with GPT-OSS-120B)	Reflection rounds: 1-10 stable
All show consistent +5–12 gains	↘ Robust to noisy reflector (+5.4 with noise every 5 steps)	Dedup. threshold: 50-90% stable
Online variant strongest in all	Only fails when adversarial every step	Pruning threshold: 10K-100K stable

Method	Generator LLM	Reflector LLM	Curator LLM	Accuracy (↑)
Base LLM	DeepSeek-V3.1	-	-	70.7
ACE	DeepSeek-V3.1	GPT-OSS-120B	DeepSeek-V3.1	76.6 (+5.9)
ACE	DeepSeek-V3.1	DeepSeek-V3.1	DeepSeek-V3.1	78.3 (+7.6)
ACE	DeepSeek-V3.1	GPT-5.1	DeepSeek-V3.1	78.5 (+7.8)

Table 16: **Weaker Reflector Models on FiNER.** We vary the Reflector while keeping the Generator/Curator fixed. Parentheses report deltas vs. the base LLM.

Harmful Reflector Frequency (every X iters)	Accuracy (↑)
base LLM	70.7
1	66.7 (-4.0)
5	76.1 (+5.4)
10	77.0 (+6.3)
25	77.8 (+7.1)
50	78.2 (+7.5)
100	78.2 (+7.5)
No harmful reflector	78.3 (+7.6)

Table 17: **Robustness to Noisy or Harmful Reflector Feedback on FiNER.** We invoke a harmful reflector once every X adaptation steps. Parentheses report deltas vs. the base LLM.

ACE's gains are ***not*** contingent on *specific model*, *reflector* strength, or fine-tuned *hyperparameters*

Discussion

Two divergent paths forward

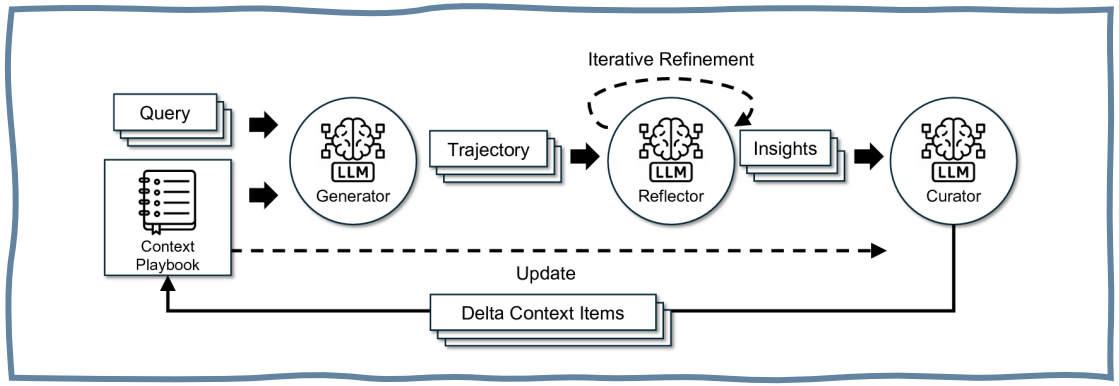
A closer look at the claims

Discussion – Future Work

Future Work: Two divergent paths forward

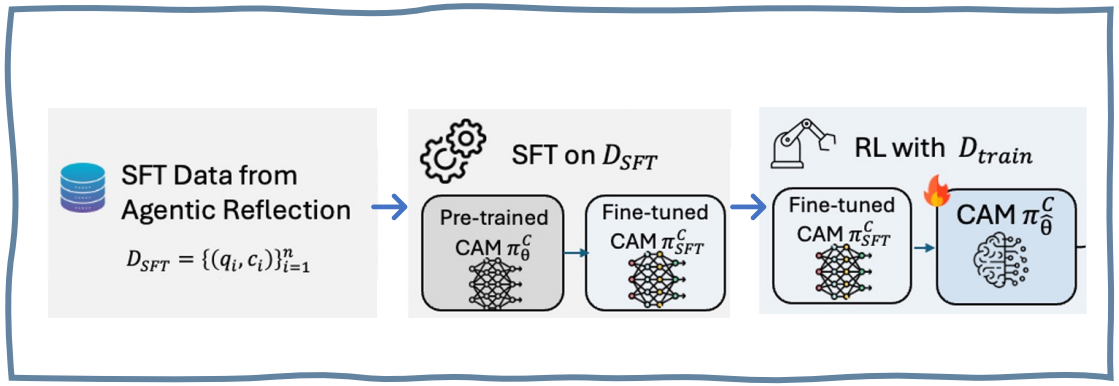
Frozen LLM, smarter playbook — or train the LLM to use context better?

Training-free path



- Refine **ACE**: Better reflectors, longer horizons
- Keep the LLM frozen, improve the playbook

Training-based path



- **CLEAR** (AWS, 2026): 32B CAM with SFT + RL
- Train an extra model — generate context *dynamically*
- Reports +6.75 over **ACE** on AppWorld

ACE assumes *inference-time compute* is cheap; CLEAR assumes *training-time compute* is cheap.

Discussion – Critical View

Critical examination of ACE itself

1 Brevity bias

Is this universal, or a specific finding repackaged as a general failure mode?

Gao, Shuzheng, et al. "*The prompt alchemist: Automated llm-tailored prompt optimization for test case generation.*" arXiv preprint arXiv:2501.01329 (2025).

2 Context Adaptation

Bundles prompt-opt (GEPA) + agent memory (DC) under one rhetorical roof. → useful for the paper 🤔

(Coined by the authors to bundle two distinct lines under one umbrella.)

3 Three-role design — under-tested

Ablation removes Reflector, but doesn't test: "*what if one LLM call handles all three at once?*"

(Is the division of labor essential, or just convenient?)

4 Heavy Reflector Dependence

Reflector quality bound ACE's ceiling → Adversarial reflector breaks ACE entirely (Section 4.67)

(acknowledged limitation)

What's solid: the ***engineering***. What's rhetorical: the ***framing***.

Contribution

1 Identifies + names two failure modes

- brevity bias and context collapse — diagnosed with empirical evidence

2 Three-role agentic architecture (Generator/Reflector/Curator)

- structured division — try (Generator), diagnose (Reflector), merge (Curator)

3 Incremental delta + grow-and-refine

- deterministic merge — the real safeguard against DC's collapse

4 Strong empirical validation

- across agent (AppWorld) + domain (Finance, …) + 4 LLM families

Thank You

Yejin Yoon

HYU NLP Lab.

Dept. of Computer Science
Hanyang University, South Korea

stillwithyou@hanyang.ac.kr