LEARNING WHAT TO LEARN: CURRICULUM CURATION FOR TEST-TIME AGENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time learning enables large language model (LLM) agents to adapt during inference without costly retraining, yet prior work largely treats test-time experience as equally useful. We ask a simple question: *what data should agents learn from at test time?* Focusing on task selection and ordering for context-based adaptation, we hypothesize that redundant or overly simple examples offer diminishing returns, while curated curricula improve sample efficiency. Using the Agentic Context Engineering (ACE) framework, we evaluate on the AppWorld benchmark featuring tool-use and coding agents. We show that careful data selection can match full-dataset performance using only $\sim$30% of training tasks, and that task ordering measurably affects learning outcomes. Our results position curriculum curation as a first-class design dimension for efficient test-time agent learning and practical deployment.

1 INTRODUCTION

Training large language models (LLMs) from scratch is computationally expensive and resource-intensive. This has motivated growing interest in *test-time learning*, techniques that adapt and improve model behavior during inference without updating parameters. For *LLM agents* (systems that interact with environments, use tools, and execute multi-step tasks), test-time learning is particularly appealing, as it enables agents to improve from experience without costly retraining. Recent approaches span context-based methods like ACE (Zhang et al., 2025a), GEPA (Agrawal et al., 2025) and Dynamic Cheatsheet (Suzgun et al., 2025), as well as reflective pipelines like Reflexion (Shinn et al., 2023) and Self-RAG (Asai et al., 2024).

While these methods demonstrate promise for agentic systems, a fundamental question remains underexplored: *are all training examples equally valuable for test-time agent learning?* In traditional supervised learning, curriculum design and data selection are known to significantly impact learning efficiency (Bengio et al., 2009; Mirzasoleiman et al., 2020; Killamsetty et al., 2021). We hypothesize that similar principles apply to test-time learning for agents, that careful selection and ordering of task samples can improve both learning outcomes and sample efficiency. Intuitively, redundant task examples introducing overlapping concepts may provide diminishing returns, while overly simple tasks may fail to provide sufficient signal for agent improvement.

To investigate this hypothesis, we conduct experiments using Agentic Context Engineering (ACE) (Zhang et al., 2025a) as our implementation framework, focusing on context-based test-time learning for agents. We evaluate on AppWorld (Trivedi et al., 2024), a challenging agent benchmark requiring tool use, code generation, and multi-turn interaction with dynamic environments. Our main findings are: **(1) Data selection matters substantially for agent learning:** we achieve comparable performance using only $\sim$30% of the full training set through strategic example selection. **(2) Ordering affects agent learning performance:** preliminary results suggest that harder examples may provide richer learning signal, though the precise mechanisms need further investigation.

These findings suggest that sample efficiency in agent test-time learning can be dramatically improved through principled curriculum curation, opening avenues for more practical deployment of adaptive LLM agents.

2 Background and Problem Formulation

2.1 Background

Test-Time Learning for Agents is a paradigm in which an LLM-based agent acquires task-relevant knowledge during inference, without updating its pre-trained weights, by leveraging experience gathered online from rich contextual signals such as documents, tool traces, execution trajectories, or interaction logs (Zhang et al., 2025a; Suzgun et al., 2025; Agrawal et al., 2025). Complementary lines of work emphasize reflective self-improvement through critique and revision (Wang et al., 2022; Shinn et al., 2023; Asai et al., 2024), as well as environment-driven adaptation in interactive settings (Shridhar et al., 2020; Wang et al., 2023; Yang et al., 2024). The resulting knowledge can be immediately applied to the current task and optionally distilled into compact, structured artifacts, such as summaries, rules, plans, or playbooks, that persist across episodes and serve as future conditioning context (Zhang et al., 2025a;b; Agrawal et al., 2025; Suzgun et al., 2025).

Curriculum Learning studies how the selection and ordering of training examples affect learning efficiency and generalization, with early work showing that structured progressions can significantly accelerate optimization and improve final performance (Bengio et al., 2009). Subsequent research casts curriculum design as data subset or coreset selection, identifying small representative sets that preserve accuracy while reducing computation (Mirzasoleiman et al., 2020; Killamsetty et al., 2021). Most prior work, however, assumes offline, gradient-based training over static datasets. In contrast, test-time learning for agents must operate under strict inference budgets and sequential adaptation, motivating new instantiations of curriculum principles when experience is distilled into context rather than model parameters.

2.2 Problem Formulation: Test-Time Contextual Optimization

We formalize *test-time contextual optimization* for LLM agents, where adaptation data is distilled into an intermediate artifact (e.g., a playbook) that conditions downstream inference. Let $D = \{e_1, \ldots, e_N\}$ denote a pool of candidate adaptation experiences. Under a fixed test-time budget, the agent selects and processes at most $k \ll N$ experiences sequentially, updating a playbook P_t via a distillation operator

$$P_t = f(P_{t-1}, e_t), \quad P_0 = \emptyset. \tag{1}$$

After k steps, the resulting playbook P_k is used to guide behavior on downstream tasks, yielding performance $M(\cdot)$. We define test-time contextual optimization as the problem of selecting a subset $S \subseteq D$ and an ordering π to maximize post-distillation performance:

$$\max_{S,\, |S| \leq k,\, \pi} M(\text{Agent}(P)), \quad \text{where } P = \text{Distill}(S, \pi). \tag{2}$$

Because playbook distillation is sequential and compressive, it is generally order-dependent:

$$\text{Distill}(\{e_i, e_j\}, (e_i, e_j)) \neq \text{Distill}(\{e_i, e_j\}, (e_j, e_i)). \tag{3}$$

3 Experiments and Results

We present two experiments that isolate the effects of *data selection* and *data ordering* for curriculum curation in test-time agent learning via the ACE (Zhang et al., 2025a) framework. Across all experiments, the agent architecture, adaptation framework, and evaluation metrics are held fixed; we vary only the selected adaptation data or the order in which it is processed. We use ACE[1] on top of a ReAct-style agent (Yao et al., 2023) and evaluate on AppWorld (Trivedi et al., 2024), a benchmark composed of tool-using, multi-step coding tasks. Our results yield two key findings:

- Test-time learning performance is driven more by *which* experiences are selected than by *how many*: small, carefully curated adaptation sets recover most of the gains of the full dataset, while redundant examples offer diminishing or even negative returns. (§3.1)
- Experience ordering shapes the procedural abstractions induced during adaptation, leading to systematic differences in robustness and generalization across task difficulty. (§3.2)

[1] We adopt a batch size of 1 and set the maximum number of Reflector refinement rounds and the maximum number of epochs in offline adaptation to 1.

Method	Selection Strategy	Train Size	Playbook Size	Test-Normal TGC↑	Test-Normal SGC↑	Test-Challenge TGC↑	Test-Challenge SGC↑	Average
Full Train Set								
ReAct	Full	90	–	63.7	42.9	41.5	21.6	42.4
ReAct + ACE	Full	90	5,874	65.5 +1.8	57.1 +14.2	51.1 +9.6	**38.1 +16.5**	53.0 +10.6
Reduced Train Set								
ReAct + ACE	Oracle*	60	4,587	67.9 +4.2	55.4 +12.5	50.1 +8.6	36.0 +14.4	52.4 +10.0
ReAct + ACE	Embedding	60	3,754	**71.4 +7.7**	**58.9 +16.0**	**53.7 +12.2**	**38.1 +16.5**	**55.5 +13.1**
ReAct + ACE	Oracle	30	1,946	66.7 +3.0	51.8 +8.9	53.2 +11.7	32.4 +10.8	51.0 +8.6
ReAct + ACE	Embedding	30	1,812	66.1 +2.4	50.0 +7.1	50.4 +8.9	32.4 +10.8	49.7 +7.3
ReAct + ACE	Embedding	10	1,031	62.5 -1.2	48.2 +5.3	49.6 +8.1	25.9 +4.3	46.6 +4.2

Table 1: **Data selection results on AppWorld. Oracle** selects one task per human-labeled task family, while **Oracle*** selects two tasks per family, introducing additional within-family redundancy. **Embedding** selects a single task per cluster based on embedding similarity.

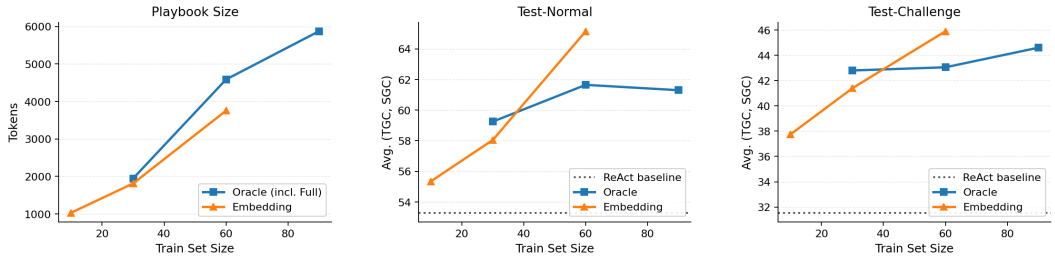

Figure 1: **Effect of data selection on playbook size and performance. Left:** Playbook size (tokens) induced by ACE as a function of training set size. **Middle/Right:** Average performance on the AppWorld Test-Normal and Test-Challenge evaluation splits. The dotted horizontal line denotes the ReAct baseline. Squares indicate oracle selection based on human-labeled task families (including the full training set), while triangles indicate embedding-based representative selection. Lines connect methods using the same selection strategy.

3.1 Results on Data Selection

In the first experiment, we investigate whether all training tasks are equally valuable for test-time learning in agents. Specifically, we study how different task selection strategies affect learning efficiency when adapting agents via ACE-style contextual playbooks on the AppWorld training set, which consists of 90 tasks organized into 30 human-annotated clusters.

Selection Strategies. We compare the following training set construction strategies: (1) *Full*: all 90 training tasks. (2) *Oracle (k/cluster)*: human-annotated selection of k tasks per cluster. (3) *Embedding (1/cluster)*: selection of one representative task per cluster using task embedding similarity; For the embedding-based selection, we use OpenAI's `text-embedding-3-large` model.

Results. Table 1 summarizes results on AppWorld. Using the full training set, ACE improves over the ReAct baseline, increasing the average score from 42.4 to 53.0. More importantly, careful data selection achieves comparable or better performance with far fewer tasks. With 60 tasks selected via embedding-based clustering (one per cluster), the agent reaches an average score of 55.5, outperforming the full 90-task setting. Even with only 30 tasks, both oracle and embedding-based selection match full-data performance, highlighting strong sample efficiency. Figure 1 further shows that embedding-based selection consistently yields more compact playbooks while achieving equal or better performance on both Test-Normal and Test-Challenge. In contrast, oracle selection exhibits diminishing returns when multiple highly similar tasks are included per cluster. Overall, prioritizing representative coverage over redundancy improves both performance and playbook efficiency, and even very small adaptation sets remain well above the ReAct baseline.

Method (Train Size)	Playbook Size	Test-Normal TGC↑	Test-Normal SGC↑	Test-Challenge TGC↑	Test-Challenge SGC↑	Average
ReAct (90)	–	63.7	42.9	41.5	21.6	42.4
Random (54)	3,440	**68.5** +4.8	**53.6** +10.7	51.3 +9.8	33.1 +11.5	**51.6** +9.2
Easy → Hard (54)	3,473	63.7 +0.0	48.2 +5.3	50.8 +9.3	31.6 +10.0	48.6 +6.2
Hard → Easy (54)	3,338	63.1 -0.6	46.4 +3.5	**54.7** +13.2	**34.5** +12.9	49.7 +7.3

Table 2: **Data ordering results on AppWorld.** Ordering effects using the same 54 training samples (18 per difficulty). Playbook size reports the total token length of the constructed playbook. Colored values indicate Δ relative to the ReAct baseline (green: improvement, red: decrease).

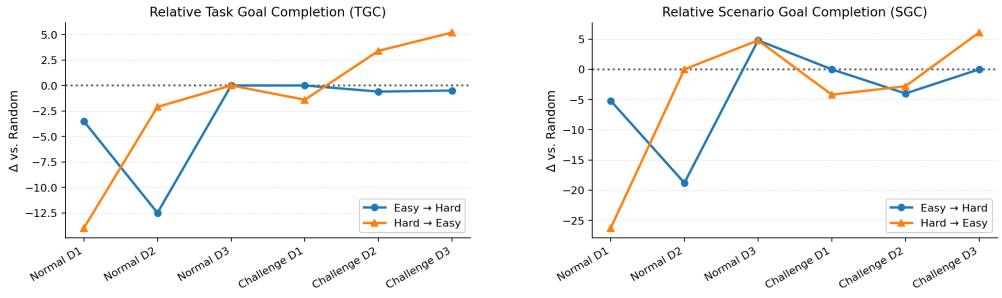

Figure 2: **Effect of data ordering on AppWorld** using a fixed 54-example adaptation set. Values show deltas relative to random ordering; dotted zero lines indicate parity with the random baseline.

3.2 Results on Data Ordering

We investigate whether the order in which tasks are processed during adaptation affects learning outcomes. To isolate ordering effects from data composition, we fix the adaptation set to the same 54 training tasks, evenly sampled across difficulty levels (18 per difficulty). We vary only the sequence in which these experiences are processed during adaptation.

Ordering Strategies. We consider three ordering strategies. *Random* randomly permutes the tasks. *Easy-to-Hard* and *Hard-to-Easy* orders tasks by increasing and decreasing difficulty.

Results. Table 2 and Figure 2 summarize the effect of data ordering using the same 54-task adaptation set. All strategies outperform the ReAct baseline, but their benefits differ by metric and split. Random ordering achieves the strongest overall gains, especially on Test-Normal, while Hard-to-Easy ordering performs best on Test-Challenge, yielding the largest improvements in TGC (+13.2) and SGC (+12.9), with gains concentrated at higher difficulty levels as shown in Figure 2. Easy-to-Hard ordering shows mixed results, helping easier tasks but hurting harder ones. Overall, ordering primarily redistributes learning signal across difficulty regimes rather than providing uniform gains.

4 Conclusion

We examine how test-time learning for LLM agents is shaped by the organization of adaptation curriculum. Through controlled experiments on AppWorld using the ACE framework, we show that agent performance depends critically on how experience is selected and ordered. We find that small, carefully curated adaptation sets can match or exceed the performance of much larger ones, and that different orderings induce distinct generalization behaviors across task difficulties.

Looking ahead, our findings motivate several directions for future research. One promising avenue is the development of automated test-time optimization methods that jointly select and order adaptation experiences under explicit context-length or computation constraints. Extending these ideas to online settings could further enable agents to dynamically curate and revise adaptation data as environments evolve, helping to develop more flexible and long-lived agentic systems.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning. *arXiv preprint arXiv:2507.19457*, 2025.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. 2024.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

Krishnateja Killamsetty, Sivasubramanian Durga, Ganesh Ramakrishnan, Abir De, and Rishabh Iyer. Grad-match: Gradient matching based data subset selection for efficient deep model training. In *International Conference on Machine Learning*, pp. 5464–5474. PMLR, 2021.

Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning*, pp. 6950–6960. PMLR, 2020.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic Cheatsheet: Test-Time Learning with Adaptive Memory. *arXiv preprint arXiv:2504.07952*, 2025.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A Controllable World of Apps and People for Benchmarking Interactive Coding Agents. *arXiv preprint arXiv:2407.18901*, 2024.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models. *arXiv preprint arXiv:2510.04618*, 2025a.

Qizheng Zhang, Michael Wornow, and Kunle Olukotun. Agentic Plan Caching: Test-Time Memory for Fast and Cost-Efficient LLM Agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2025b.

A PLAYBOOK ANALYSIS

To understand how experience ordering affects performance, we analyze the learned playbooks induced by different ordering strategies. Although all methods use the same adaptation set, ordering systematically shapes the procedural abstractions that are learned: Easy-to-Hard induces highly prescriptive, schema-dependent rules that excel on clean tasks but are brittle under shift; Random produces more general-purpose, coverage-oriented heuristics with strong average performance; and Hard-to-Easy yields robustness-focused rules that emphasize retries and failure handling, which aligns with its strength on challenging tasks. Overall, experience ordering primarily influences which abstractions are learned, rather than overall competence.

A.1 VERBATIM PLAYBOOK RULES

Easy-to-Hard Ordering. Playbooks learned under Easy-to-Hard ordering contain a high density of absolute and schema-dependent rules. Representative examples include:

- `Always verify the exact parameter names in API documentation rather than assuming naming consistency across similar operations.`
- `Never assume the first page contains all results---check API documentation for pagination parameters and handle them systematically.`
- `Always retrieve detailed metadata using the authoritative API before performing any filtering or comparison.`

These rules encode narrow execution assumptions that reduce ambiguity but can over-specialize to clean task structures.

Random Ordering. Random ordering induces broader, less brittle playbook rules that emphasize coverage and adaptability:

- `Systematically collect data from all relevant sources before performing comparative analysis.`
- `When automated parsing fails, visual inspection of the output can be a valid fallback strategy.`
- `Verify final state rather than relying solely on intermediate success messages.`

These heuristics allow the model to flexibly adapt to variation in task structure without committing to a single procedural template.

Hard-to-Easy Ordering. Playbooks learned under Hard-to-Easy ordering contain rules that explicitly reason about ambiguity, retries, and idempotency:

- `Recognize that certain API errors (e.g., already exists) may indicate successful completion rather than failure.`
- `Treat idempotent operation errors as confirmation of the desired state instead of retrying indefinitely.`
- `After irreversible operations, always verify completion by checking final state rather than assuming success.`

Such rules are particularly important for long-horizon and failure-prone tasks.

A.2 QUANTITATIVE PLAYBOOK STRUCTURE ANALYSIS

We further analyze playbook structure using simple lexical statistics that capture rule style rather than semantics. Table 3 reports the frequency of selected indicators across ordering strategies.

Ordering	Number of Rules	"always"	"never"	"verify"	"fallback"	"retry"
Easy → Hard	51	33	6	17	6	0
Random	60	23	0	19	1	0
Hard → Easy	62	17	3	24	1	3

Table 3: **Lexical statistics of learned playbooks.** Counts reflect frequency of selected keywords and indicate differences in rule rigidity and robustness orientation.

We observe clear differences in the number of rules in the learned playbooks across ordering strategies. Counting all distilled playbook entries—including strategy rules, verification checks, troubleshooting heuristics, and API-specific guidance—Easy-to-Hard ordering produces the most compact playbook in terms of rule count, while Random ordering yields a larger set of rules. Hard-to-Easy ordering results in the largest number of rules, which may reflect reduced early consolidation and the accumulation of additional, often case-specific, heuristics. Notably, these differences arise despite comparable playbook token lengths across orderings. This indicates that experience ordering primarily affects the structure and granularity of learned rules, rather than overall context length.

Furthermore, Easy-to-Hard ordering produces the highest density of absolute constraints, which may reflect over-specialized procedural assumptions. Random ordering yields fewer absolute rules and maintains a balance between verification and flexibility, consistent with its strong in-distribution performance. Hard-to-Easy ordering emphasizes verification and is the only strategy that consistently induces retry- and failure-aware rules, which aligns with its improved robustness on challenging tasks. These results support the hypothesis that experience ordering shapes the nature of induced playbook rules and mediates the trade-off between generality and specialization.

B The Use of Large Language Models (LLMs)

This work focuses on developing algorithms and system frameworks for effective context adaptation in large language models (LLMs). Accordingly, our experiments employ LLMs for the empirical evaluation of the proposed methods. For paper preparation, we used LLMs only to polish writing (e.g., correcting grammatical errors), and not to generate new text from scratch.