Improving Language Agents through BREW: Bootstrapping expeRientially-learned Environmental knoWledge

Anonymous Author(s)
Affiliation
Address
email

Abstract

Large Language Model (LLM)-based agents are increasingly applied to tasks requiring structured reasoning, tool use, and environmental adaptation, such as data manipulation, multistep planning, and computer-use automation. However, despite their versatility, current training paradigms for model weight optimization methods, like PPO and GRPO, remain relatively impractical with their high computational overhead for rollout convergence. In addition, the resulting agent policies are difficult to interpret, adapt, or incrementally improve. To address this, we investigate creating and refining structured memory of experiential learning of an agent from its environment as an alternative route to agent optimization. We introduce **BREW** (Bootstrapping expeRientially-learned Environmental knoWledge), a framework for agent optimization for downstream tasks via KB construction and refinement. In our formulation, we introduce an effective method for partitioning agent memory for more efficient retrieval and refinement. BREW uses task graders and behavior rubrics to learn insights while leveraging state-space search for ensuring robustness from the noise and non-specificity in natural language. Empirical results on real world, domain-grounded benchmarks – OSWorld and τ^2Bench – show BREW achieves $10-20\%$ improvement in task precision, $10-15\%$ reduction in API/-tool calls leading to faster execution time, all while maintaining computational efficiency on par with base models. Unlike prior work where memory is treated as static context, we establish the KB as a modular and controllable substrate for agent optimization – an explicit lever for shaping behavior in a transparent, interpretable, and extensible manner.

1 Introduction

Large Language Model (LLM) based agents are rapidly being deployed for structured reasoning, tool use, and autonomous interaction in real-world environments [16]. From computer-use and spreadsheet automation to software engineering pipelines, these agents drive tasks such as multi-step planning, data manipulation, and adaptive workflows [21, 13, 32, 2, 19]. For example, a language agent might help automate a multi-step workflow like collecting data from different sources, cleaning or validating it, and then uploading it onto a dedicated server, all while adjusting its plan if the format or structure of the data changes unexpectedly [31, 35, 25, 3]. Yet, despite these successes, top-performing agents generally score underwhelmingly on challenging real-world benchmarks—well behind human experts, who routinely exceed 70% success rates [34, 4, 27, 18]. As an example, consider the following scenario:

Submitted to Multi-Turn Interactions in Large Language Models Workshop @ NeurIPS 2025

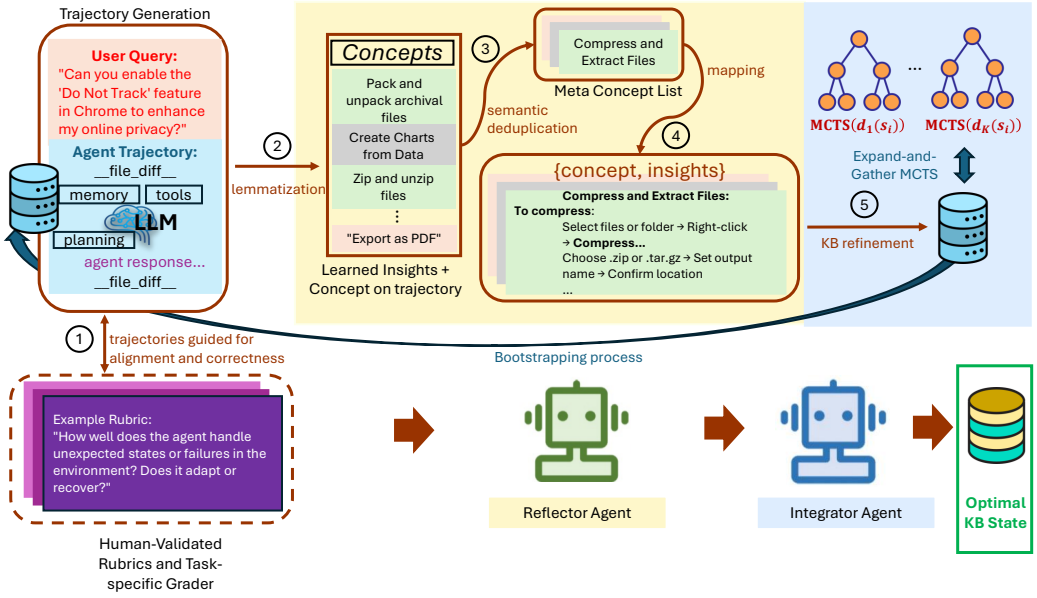

Figure 1: BREW architecture overview using examples from the OSWorld dataset. Step 1 indicates the trajectory generation process with agent alignment to human-validated rubrics and correctness using task-specific grader. Steps 2–4 indicate the Reflector Agent, which learns key concepts and corresponding insights from trajectories. Step 5 indicates the Integrator Agent, which integrates knowledge from the Reflector Agent to bootstrap the KB. We introduce Expand-and-Gather MCTS to further find the best KB configuration as the KB is iteratively refined through reward-guided optimization.

> A computer-use agent in an Ubuntu environment tasked with automating software installation across multiple sessions. In its first encounter, it struggles through a 47-step process: opening the wrong package manager, executing redundant dependency checks, and making 23 API calls to complete what could be a 6-step workflow. When presented with a similar installation task in the next session, the agent repeats the same inefficient exploration – *as if encountering the problem for the first time.* A human user, by contrast, would likely have a recollection from internalized memory of the optimal sequence after the first attempt, recognizing the environmental patterns and tool combinations that lead to success.

This scenario illustrates a fundamental limitation of current language agents: despite their impressive capabilities in reasoning and tool use, they lack the ability to accumulate and apply experiential knowledge across task sessions. Each interaction begins from a blank slate, forcing agents to repeatedly explore the same action spaces and rediscover the same solutions [9]. Real-world tasks like long horizon multi-stage automation demand more than just "reactive" [33] tool loops. They require persistent & interpretable learnings from past experiences - what works, what fails and why.

To close this gap, recent work has explored learning agent behavior using model weight optimization [23, 22, 24], where agents are trained to maximize success across a wide variety of tool-use episodes. However, while conceptually sound, this suffers from practical limitations. First, it requires expansive exploration over large rollout spaces to converge, especially in domains where tasks are diverse, goals are sparsely defined, and intermediate feedback is noisy or delayed. Second, the resulting policies are often opaque—difficult to interpret, revise, or debug—limiting their real-world deployability. Finally, these policies are tightly coupled to the task distributions they were trained on, making it difficult to adapt or incrementally improve them when downstream requirements shift.

In contrast, others have explored learning of knowledge onto a memory module that remains attached to an agent. These existing memory-augmented agents can be broadly classified into either ones which (i) store only transient trajectory contexts that vanish between episodes like Mem0 [7, 29], or (ii)

embed high-level notes directly in the prompt such as MetaReflection [10] and GEPA[1]. While the latter often do not retain actionable details for future simple tasks, neither of these approach supports modular updates, fine-grained retrieval, or transparent inspection of what the agent "knows." [28].

Leveraging learnings from both camps, we introduce BREW (**B**ootstrapping expe**r**ientially-learned **e**nvironmental kno**w**ledge), a framework that incrementally constructs and refines a knowledge base (KB) a structured collection of concept-level documents in natural language, directly from an agent's past interactions. This KB then serves as a persistent memory for the agent to retrieve knowledge in future executions to improve precision and efficiency outcomes. Our key contributions are–

- *Novel experience-driven KB construction.* We propose a technique for leveraging agent's past interaction trajectories to generate uniquely-partitioned concept-level KB documents. This process is guided by rubrics and task-specific graders which ensures that memories are both semantically aligned with task objectives and human-interpretable.
- *State-space search for memory optimization.* We formalize the selection and update of KB entries as a state search problem and introduce an efficient reward-guided learning scheme, Expand-and-Gather Monte Carlo Tree Search (EG-MCTS), that learns to prioritize the most impactful memories for robust, multi-step reasoning.
- *State-of-the-art results.* On domain-grounded benchmarks including OSWorld and τ^2Bench, BREW achieves significant gains of in the range of $10 - 20\%$ towards task precision as well as $10 - 15\%$ fewer steps leading to faster execution, while maintaining memory and compute costs comparable to base LLMs.

2 Preliminary & Related Work

Agent Learning from Demonstrations Recent work has leveraged LLMs to isolate reusable skills through interactive decomposition: one method distills sub-goals from expert trajectories into hierarchical planning and execution policies [11], and another synthesizes executable functional abstractions for advanced mathematical reasoning via program induction [14]. These approaches focus on structured skill extraction from LLM-guided interactions, yet remain reliant on static decomposition or offline synthesis. In contrast, BREW dynamically constructs and refines an experiential memory—learning necessary semantic fragments via rollout-generated insights and structured knowledge-base search (MCTS)—to support long-horizon, memory-augmented planning.

Agentic Memory The concept of providing agents with controllable memory has a rich history. [17]. Memory mechanisms are attracting more and more attention lately [20, 26, 28, 7, 30, 12]. These works focus towards storing relevant context in a structured format like graph or a tree so as to RAG over it. Despite their effectiveness these methods perform well for most cases. However, when the queries are ambiguous, requires multi-hop reasoning and long range comprehension these techniques struggle to perform the tasks [12]. In contrast to prior works BREW uses a state search to explore possible memory states. This allows BREW to select the memory state where the reward during exploration is highest making it more robust to ambiguous queries and long range comprehension. We employ MCTS [8] as a state search algorithm to explore the potential states of the memory by expanding to new and potentially different states of memory based on same interactions. We discuss the state search process more formally in Section 3.3.

3 BREW: Architecture

This section describes our proposed **B**ootstrapping expe**R**ientially-learned **E**nvironmental kno**W**ledge model, BREW, which constructs and iteratively refines a KB using trajectory insights guided by human-validated general-purpose agent behavior metrics, task-specific evaluation, and latent insight generation. We introduce a novel decomposition the problem of learning the optimal KB by partitioning memory as local documents associated with semantic concepts, and formulate the KB learning problem as a state space search by proposing Expand-and-Gather Monte Carlo Tree Search (EG-MCTS). Figure 1 provides an architecture overview of BREW, and Algorithm 1 describes pseudocode.

3.1 Trajectory Generation

Given the training dataset, we generate full-length trajectories, hereby referred to as rollouts, for each query using an LLM-powered agent conditioned on its associated KB. At initialization, the KB is empty, and the LLM is used with a decoding temperature of 0 to ensure deterministic behavior. Further details on training and test splits are in Experiments Section. Each rollout is evaluated using a correctness grader, which assigns a binary label: *success* or *failure* and a qualitative rubric assessment against a set of human-validated general-purpose agent behavior rubrics [6] (Step 1 in Figure 1).

3.2 Reflector and Integrator Agents

Reflector Agent: `ReflAgent` takes as input a rollout with its rubric and correctness labels, and outputs sentence-level insights with mapped concepts:

$$\{\textit{concepts}, \textit{insights}\} = \texttt{ReflAgent}(\{\textit{rollout}, \textit{eval}\}). \tag{1}$$

Examples of concept–insight pairs appear in Step 2 of Figure 1.

Concept Deduplication: Concept–insight pairs are annotated independently per rollout, often producing overlapping or paraphrased concepts. We address this via semantic clustering (Steps 3–4, Figure 1; Algorithm 1, line 3): contextual embeddings for each concept are generated using an LLM, clustered, and each insight is mapped to its cluster representative. Details appear in Algorithms **??** and **??** in Appendix **??**.

Integrator Agent: `IntegAgent` incrementally builds and refines KB documents $\{d(s_i)\} \in \mathcal{D}(s_i)$ during environment interaction. Instead of a centralized memory, the KB is partitioned into local documents, each tied to a meta concept. This design enables (1) efficient, context-specific retrieval; (2) modular updates with minimal interference; and (3) natural alignment with task semantics, as deduplicated meta concepts capture meaningful behavioral abstractions. Unlike prior work assuming flat memory or dialogue histories, this structure is well-suited for long-horizon, procedural tasks where behaviors cluster around discrete skills.

The KB is dynamically populated: concepts central to the dataset receive more updates, shaping memory around frequent behaviors. At each state, for meta concept k, `IntegAgent` updates its document d_k via

$$d_k(s_{i+1}) \leftarrow \texttt{IntegAgent}(k, \textit{insights}_k, d_k(s_i)). \tag{2}$$

To reduce LLM variance and improve consistency, we use the Expand-and-Gather MCTS (EG-MCTS) method (Figure 2).

Formally, the KB at state s_i is the union of all concept-localized documents:

$$\mathcal{D}(s_i) = \bigcup_{k \in \mathcal{K}} \{d_k(s_i)\}, \tag{3}$$

where $\mathcal{K}$ is the set of all meta concepts and $d_k(s_i)$ is the document for concept k at state s_i.

3.3 Expand-and-Gather MCTS for Optimal KB Search

We start by creating a set of meta-concepts after deduplicating concepts extracted by `ReflAgent` using the first set of trajectory rollouts. We freeze this meta-concept set $\mathcal{K}$, and use it to initialize a KB with an empty document per concept $k \in \mathcal{K}$.

We model the problem of finding the optimal KB $\mathcal{D}^*$ as a search problem in the *state* space of all possible KBs $\mathcal{D}$. To simplify this state search, we model KB $\mathcal{D}$ as a collection of concept level documents. This modeling allows us to break down the larger search space into a collection of simpler document level search problems for each concept k to find the optimal document d_k^*. We then construct the optimal KB $\mathcal{D}^*$ by combining all optimal documents d_k^* for each concept k as follows:

$$\mathcal{D}^* = \bigcup_{\forall k} \{d_k^*\} \tag{4}$$

Notably, even though we are modeling document level search as independent optimization problems, each document in the KB is *not* independent of the others. For example, an agent can retrieve any

Algorithm 1 BREW: Bootstrapping Experientially-learned Environmental Knowledge

Require: Training samples $\mathcal{Q}_{\text{train}}$, eval samples $\mathcal{Q}_{\text{eval}}$, rubrics, iterations M, candidates per expansion h

Ensure: Optimized KB $\mathcal{D}^*$

Initialization

$\mathcal{D}_0 \leftarrow \varnothing$
$\mathcal{B} \leftarrow \text{GENERATEINSIGHTS}(\mathcal{Q}_{\text{train}}, \mathcal{D}_0, \text{rubrics})$
$\mathcal{K} \leftarrow \text{DEDUPLICATECONCEPTS}(\mathcal{B})$ ▷ Initial concept set
for each $k \in \mathcal{K}$ **do**
  $d_k^0 \leftarrow \text{INTEGAGENT}(k, \mathcal{I}_k, \varnothing)$
  Initialize tree_k with root node d_k^0
end for
$\mathcal{D}_{\text{current}} \leftarrow \bigcup_{k \in \mathcal{K}} \{d_k^0\}$ ▷ Initial KB

EG-MCTS Optimization

for $t = 1$ to M **do** ▷ Parallel expansion across concepts
  for each $k \in \mathcal{K}$ **do**
    $s_k \leftarrow \text{SELECTBESTNODE}(\text{tree}_k)$ ▷ UCT selection
    $\mathcal{D}_{\text{best}} \leftarrow \bigcup_{k' \in \mathcal{K}} \{d_{k'}^{\text{best}}\}$ ▷ Current best docs
    $\text{EXPANDNODE}(s_k, k, h, \mathcal{D}_{\text{current}}, \mathcal{D}_{\text{best}}, \text{tree}_k)$
  end for ▷ Update current best documents
  for each $k \in \mathcal{K}$ **do**
    $d_k^{\text{best}} \leftarrow$ best document in tree_k
  end for
  $\mathcal{D}_{\text{current}} \leftarrow \bigcup_{k \in \mathcal{K}} \{d_k^{\text{best}}\}$
end for
return $\mathcal{D}_{\text{current}}$

Time Complexity: $O(|\mathcal{Q}_{\text{train}}| \cdot T_{\text{LLM}} + M \cdot |\mathcal{K}| \cdot h \cdot T_{\text{agent}})$

document in the KB during inference and this retrieval making it hard to assess the impact of changing a document in isolation. To solve this we propose Expand-and-Gather MCTS (EG-MCTS), which enables searching these disjoint state spaces concurrently using parallel MCTS explorations that are synced after each iteration. To achieve this we perform node expansions in the respective search spaces independently but condition reward calculation and insight generation on a running optimum KB state. Each iteration of EG-MCTS can be broken down two phases:

Expand Phase: During this stage, for each search tree, we pick the *best* state s^* and expand it concurrently. To perform this expansion the KB $\mathcal{D}(s^*)$ is constructed by including the *current document* $d_k(s^*)$ and the *best (oracle) documents* $\{d_i^*\}_{i \neq t}$ for all other positions. Thus, the KB at iteration t, $0 \leq t \leq E$ is defined as:

$$\mathcal{D}_t = d_t \cup d^*_{i:i \neq t} \tag{5}$$

We use this KB $\mathcal{D}(s_i)$ to generate trajectory rollouts which are consumed by the `ReflAgent` to generate insights. We then use the `IntegAgent` to generate various updated variants of d_k^* e.g., $d_k(s_i), ..., d_k(s_j)$, where $0 \leq i \leq E$ and $0 \leq j \leq E$. We then estimate a reward R for each of these newly generate states and update rewards of parent states using backpropagation.

Gather Phase: During this stage, the current best states from each document's MCTS tree are *gathered* together and distributed to every MCTS tree for reward calculation. This is important to 1. Estimate rewards for each expanded state, and 2. Generate new insights for further node expansion.

3.4 Reward-Guided Optimization

This section describes BREW's joint reward and loss optimization for learning an optimal KB.

Reward Objective: Each document state is rewarded based on two complementary criteria: *(i)* how well the current document contributes to **accurate downstream reasoning**, and *(ii)* how **retrievable** it is in the context of a growing KB. Formally, the total reward at time step t is defined as:

$$R_t = \lambda_{\text{corr}} \cdot R_t^{\text{corr}} + \lambda_{\text{ret}} \cdot R_t^{\text{ret}} \tag{6}$$

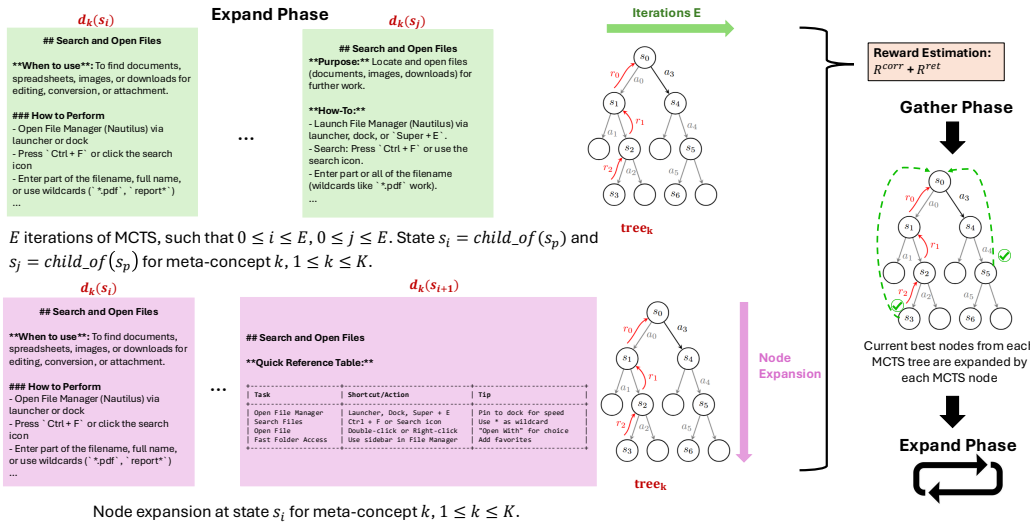

Figure 2: Illustration of BREW's KB optimization process using Expand-and-Gather MCTS with OSWorld examples. In the **Expand Phase**, for each document k, we sample the best node from tree$_k$ using UCT and perfrom node expansion. Node rewards are estimated based on correctness and retrievability. In the **Gather Phase**, the current best nodes from each tree are gathered per node, and the objective function is optimized. The process is repeated during the next iteration of KB refinement.

where R_t^{corr} is the **correctness reward**, R_t^{ret} is the **retrieval reward**, and $\lambda_{\text{corr}}, \lambda_{\text{ret}} \in [0, 1]$ are scalar weights with $\lambda_{\text{corr}} + \lambda_{\text{ret}} = 1$.

Correctness Reward: The correctness reward R_t^{corr} evaluates the accuracy of the agent's output over a held-out query set $\mathcal{Q}$, when reasoning over the current KB $\mathcal{D}_t$. It is defined as:

$$R^{\text{corr}}(d_t|\mathcal{D}_t) = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \text{Eval}_{\text{task}}(q, \texttt{agent} \oplus \mathcal{D}_t) \tag{7}$$

where $\text{Eval}_{\text{task}}$ is a task-specific evaluation function (e.g., question-answering accuracy, entailment correctness), and $\texttt{agent} \oplus \mathcal{D}_t$ denotes the agent acting over the hybrid KB.

Retrieval Reward: The retrieval reward R_t^{ret} measures how effectively the current document d_t can be retrieved from the current KB $\mathcal{D}_t$. For a held-out query set $\mathcal{Q}$, it is computed using the mean reciprocal rank (MRR):

$$R^{\text{ret}}(d_t|\mathcal{D}_t) = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \text{MRR}_q(d_t, \mathcal{D}_t) \tag{8}$$

This encourages documents that are not only helpful in reasoning but also easily retrievable via the retrieval model over $\mathcal{D}_t$.

4 Experimental Setup

Datasets We evaluate BREW on three diverse benchmarks testing different aspects of interactive agent capabilities: OSWORLD for computer-use automation [27], τ^2-Bench for tool use [5], and SPREADSHEETBENCH for data manipulation [18].

1. **OSWorld:** This benchmark tests multimodal agents on real-world computer tasks across 10 applications. We use *GTA1-7B*, a state-of-the-art computer-use agents with BREW. Tasks are evaluated using 134 custom scripts that verify final application states.
2. **τ^2-Bench:** This benchmark evaluates conversational agents on multi-turn tool-use scenarios across *Telecom*, *Retail*, and *Airline* domains. We test `o4-mini`-based tool-calling agent, constructing BREW KBs for every domain.

3. **SpreadsheetBench:** This benchmark evaluates agents on real-world spreadsheet manipulation, spanning both cell-level and sheet-level tasks. It contains 912 authentic user instructions paired with 2,729 test cases ($\tilde{3}$ per instruction), sourced from Excel forums and blogs. Spreadsheets include diverse formats with multi-table sheets (35.7%) and non-standard tables (42.7%). We test `o4-mini` using a Python tool-calling agent, and enhance it with by adding an embedding based Retrieval over the BREW KB generated over a small held-out train set of 30 samples.

Baselines We compare BREW against two widely used experiential memory approaches, *Cognee*[1] and *Agent-Mem* [30], both of which serve as established baselines for AI memory evaluation. Cognee is an open-source AI memory engine that employs a graph-plus-vector memory architecture through an Extract–Connect–Learn pipeline, enabling agents to construct cross-document and cross-context connections entirely from previously available trajectories. In contrast, Agent-Mem provides a scalable memory layer for dynamically extracting and retrieving information from conversational data, with enhanced variants incorporating graph-based memory representations. While Cognee primarily emphasizes cross-document relational reasoning, Agent-Mem focuses on scalable personalization for conversational agents.

Other Experimental Configs: For all experiments, we use `GPT-4.1-2025-04-14` as the base LLM with expansion width $e = 3$, max depth $k = 3$, and balanced reward weights $\lambda_{\text{corr}} = \lambda_{\text{ret}} = 0.5$. During MCTS node selection, we use the UCT [15] for balancing exploration and exploitation Full experimental details are provided in the Appendix.

5 Analysis & Discussion

In this section, we present findings from our evaluation of BREW. For more details on qualitative insights and discussion you may refer to the supplementary material.

Variations with State Search Strategy BREW performs a search across possible KB states using MCTS. We compare different state search strategies to determine the relative trade-offs:

1. *Iterative Refinement*: In this strategy we generate one version of each document to generate an initial KB, followed by a round of evals. We then use the aggregator agent to refine the documents over the newly learned insights. We repeat this step multiple times up to a maximum number of refinements. Note that in contrast to MCTS, in this strategy we *do not* perform node expansions and rather explore a path in the search tree.
2. *Greedy Search*: In this strategy we greedily pick the best state during each node expansion and only explore the sub-tree within it. This is in contrast to MCTS where, we explore different states using the UCT algorithm that balances exploration and exploitation.

Table 1 presents how MCTS achieves consistent performance gains across all benchmarks. These represent 1-5% improvements over alternative search strategies across tasks. Iterative refinement's poor performance reveals core limitations in the integrator agent feedback incorporation- which can be attributed to inherent stochasticity in LLMs. This makes state exploration especially important for textual optimization tasks like ours. We present a detailed analysis on how varying MCTS parameters result in different final states in appendix.

5.1 Trends across Sub-Tasks

BREW learns recipes from sub-trajectories in OSWorld. Figure 3 shows that BREW(BREW) improves success rates in 5 out of 10 OSWorld categories, achieving absolute gains of 4–16% while maintaining performance parity in the remaining categories (Chrome, Gimp, LibreOffice Calc, LibreOffice Impress, OS). The largest improvements appear in text-processing applications (LibreOffice Writer: $14\% \rightarrow 24\%$, Thunderbird: $38\% \rightarrow 54\%$) and multimedia tools (VLC: $20\% \rightarrow 27\%$), with moderate gains in multi-application and development environments. Even in settings with limited improvements in task correctness, BREWconsistently reduces execution length by 14–23 steps, highlighting more efficient planning.

[1] github.com/topoteretes/cognee

Method	OSWorld GTA1-7B	τ^2 Bench o4-mini	SpreadsheetBench o4-mini
Baseline	44.20	56.63	44.30
Cognee	46.70	57.71	42.10
Agent-Mem	43.83	52.69	42.00
BREW-Iterative	46.13	57.34	42.98
BREW-Greedy	45.55	59.14	45.94
BREW-MCTS	**47.56**	**59.14**	**46.80**

Table 1: Comparison of models under different evaluation setups, including Baseline model and BREW augmented model. We report task success rate for OSWorld, ratio of independent tasks that succeeded for τ^2 Bench, and the 1st test case pass rate for SpreadsheetBench.

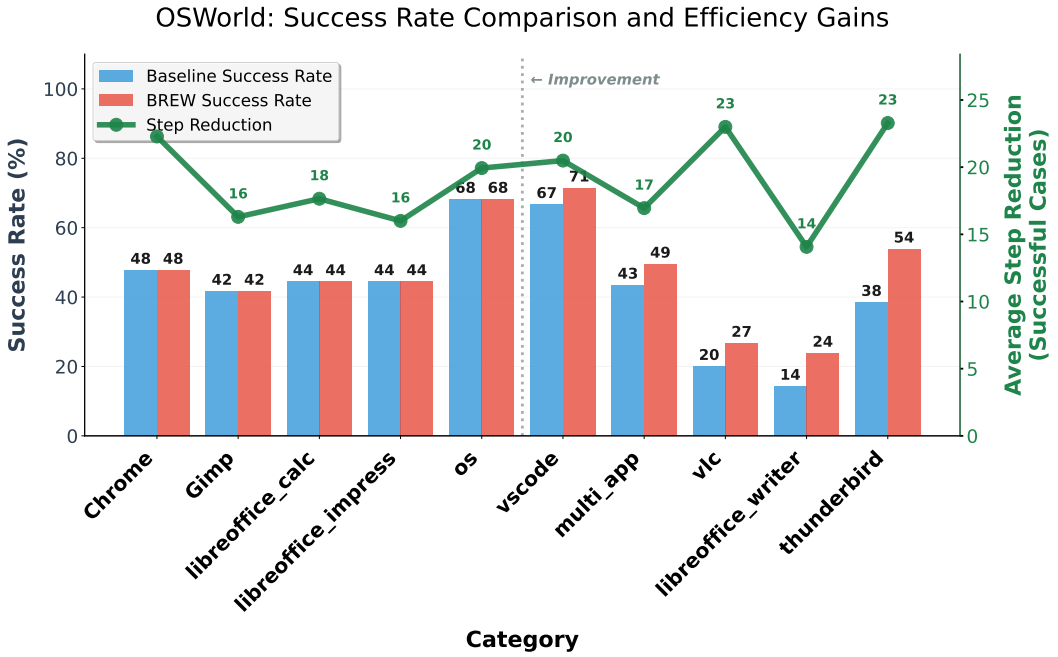

Figure 3: The bar plot represents the category-wise success rate over various tasks in the OSWorld dataset over the GTA1-agent, whereas the line plot demonstrates the reduction in the number of steps for the successful cases. Note that even in scenarios where the KB doesn't help increase the success rate, it significantly reduces the number of steps needed to succeed.

This pattern suggests that BREW's architectural enhancements are particularly effective for tasks requiring complex sequential reasoning and inter-application coordination, while preserving baseline robustness in domains constrained by intrinsic task complexity.

A qualitative analysis of the knowledge bases (KBs) constructed by BREWfurther supports this finding. We observe that BREWcaptures and represents sub-trajectory characteristics in *natural language*, including application shortcuts, standard operating procedures, and strategies for localizing UI elements. Since many UI tasks share common sub-trajectories, this representation facilitates knowledge transfer across tasks within the same application. Moreover, BREWsubstantially reduces reliance on granular UI interactions: while the baseline GTA1 model executes approximately 19,000 clicks and 17,821 keyboard actions, BREWsignificantly decreases this interaction complexity.

BREW learns aggressive resolution strategies for $\tau^2 - Bench$ To evaluate robustness of BREW, we analyzed the distribution of failure modes across the τ^2–retail dataset, focusing on four key error categories: *Wrong Argument*, *Wrong Info*, *Wrong Decision*, and *Partially Resolve*. Figure 4 presents a comparative chart for the baseline, BREW, Cognee and Agent-Mem[30].

Overall, BREW demonstrated consistent improvements across most error types compared to the baseline and competing approaches. Specifically, BREW showed a **notable reduction in "Wrong Argument" and "Wrong Decision" errors**, indicating that it was better at capturing logical dependencies in retail dialogues and making accurate decisions.

Interestingly, *Partially Resolve* errors were slightly higher for BREW than for Cognee, likely because BREW attempted more aggressive resolution strategies that occasionally failed to fully satisfy user queries. Cognee appears to capture *richer factual details* given its relatively lower *Wrong Info* errors, whereas Agent-Mem excels in *tracking conversation state and decision accuracy*, as reflected in its reduced *Wrong Decision* failures.

Improvements in Task Efficiency We observe that overall, BREWenables agents to come to a correct response quicker.

OSworld. Figure 3 demonstrates that BREW enables GTA1 to complete tasks more efficiently. Compared to the baseline GTA1 model's average of $\sim$75 steps, the BREW-augmented model completes tasks 14% faster with an average of $\sim$64 steps. Analyzing performance by outcome reveals that while step counts remain unchanged for failed cases, successful completions show a substantial 39% (rel.) reduction in execution steps, indicating improved planning efficiency for achievable tasks.

τ^2Bench. Similarly, BREW reduces average conversation turns from 29.47 to 28.43 (-3.5%), while maintaining consistent step reductions across categories. Step reductions average 1.7 steps for Retail and Telecom, but 3.1 steps for Airline, indicating greater efficiency gains in complex domains. Qualitative analysis seconds these numbers showing how knowledge base integration enables more direct task completion paths and improved planning quality, though multi-turn interactions remain necessary for complex sub-tasks.

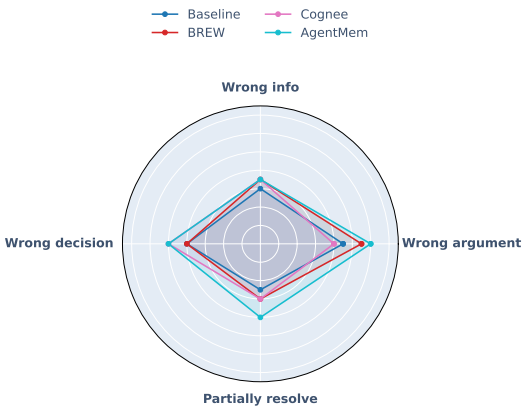

Figure 4: Distribution of errors in τ^2 Bench Retail

SpreadsheetBench. While we observe a slight increase in the number of turns across the entire benchmark suite ($4.5 \rightarrow 5.4$) in the case of the baseline versus BREW, an interesting pattern emerges in more than 82% of the cases the baseline and the BREW appended agent performs similarly with similar turn consumption. BREW leads to an improvement in 12% of the cases where the KB is able to address gaps in the baseline technique to enable the agent to go exploring further leading to positive outcomes with an average of 1 step increase in the interactions.

6 Conclusions

In this work, we explored an alternative approach to agent optimization by focusing on experiential knowledge retention rather than direct model fine-tuning. We introduced BREW, a framework that aims to construct and refine a structured, interpretable knowledge base from past agent interactions. By decomposing agent memory into concept-level documents and applying a state-search optimization strategy, BREW provides a modular and transparent substrate for memory formation. Our evaluations across OSWorld and τ^2Bench benchmarks suggest that such structured memory can support measurable improvements in task success and efficiency, while maintaining manageable computational costs. Although the observed gains are promising, we recognize that BREW's effectiveness is influenced by the quality and coverage of its training data. Future work could explore more adaptive and domain-general memory refinement techniques, as well as tighter integrations with ongoing agent planning. Ultimately, we hope this study encourages further investigation into more interpretable, memory-driven approaches to language agent development—especially in real-world environments where long-term consistency and adaptability are essential.

References

[1] Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, July 2025.

[2] Anthropic. Introducing computer use, a new Claude 3.5 Sonnet, and Claude 3.5 Haiku, October 2024. URL `https://www.anthropic.com/news/3-5-models-and-computer-use`. Accessed: 2025.

[3] Yasharth Bajpai, Bhavya Chopra, Param Biyani, Cagri Aslan, Dustin Coleman, Sumit Gulwani, Chris Parnin, Arjun Radhakrishna, and Gustavo Soares. Let's fix this together: Conversational debugging with github copilot. In *2024 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1–12, 2024. doi: 10.1109/VL/HCC60511.2024.00011.

[4] Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL `https://arxiv.org/abs/2506.07982`.

[5] Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL `https://arxiv.org/abs/2506.07982`.

[6] Param Biyani, Yasharth Bajpai, Arjun Radhakrishna, Gustavo Soares, and Sumit Gulwani. Rubicon: Rubric-based evaluation of domain-specific human ai conversations. In *Proceedings of the 1st ACM International Conference on AI-Powered Software*, AIware 2024, page 161–169, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706851. doi: 10.1145/3664646.3664778. URL `https://doi.org/10.1145/3664646.3664778`.

[7] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

[8] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Proceedings of the 5th International Conference on Computers and Games (CG 2006)*, pages 72–83. Springer, 2006. doi: 10.1007/978-3-540-75538-8_7.

[9] Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks. *The Forty-Second International Conference on Machine Learning*, 2025.

[10] Priyanshu Gupta, Shashank Kirtania, Ananya Singha, Sumit Gulwani, Arjun Radhakrishna, Gustavo Soares, and Sherry Shi. MetaReflection: Learning instructions for language agents using past reflections. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8369–8385, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.477. URL `https://aclanthology.org/2024.emnlp-main.477/`.

[11] Maryam Hashemzadeh, Elias Stengel-Eskin, Sarath Chandar, and Marc-Alexandre Cote. Subgoal distillation: A method to improve small language agents, 2024. URL `https://arxiv.org/abs/2405.02749`.

[12] Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in llm agents via incremental multi-turn interactions, 2025. URL `https://arxiv.org/abs/2507.05257`.

[13] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=VTF8yNQM66`.

[14] Zaid Khan, Elias Stengel-Eskin, Archiki Prasad, Jaemin Cho, and Mohit Bansal. Executable functional abstractions: Inferring generative programs for advanced math problems. 2025.

[15] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, pages 282–293, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-46056-5.

[16] Xinzhe Li. A review of prominent paradigms for llm-based agents: Tool use, planning (including rag), and feedback learning. In *Proceedings of the 31st International Conference on Computational Linguistics (COLING)*, pages 9760–9779, Abu Dhabi, UAE, 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.652.

[17] Michael L. Littman. An optimization-based categorization of reinforcement learning environments. 1993. URL https://api.semanticscholar.org/CorpusID:17988064.

[18] Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. *Advances in Neural Information Processing Systems*, 37:94871–94908, 2024.

[19] OpenAI. Introducing Operator, January 2025. URL https://openai.com/index/introducing-operator/. Accessed: 2025.

[20] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2024. URL https://arxiv.org/abs/2310.08560.

[21] Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

[22] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model, 2024. URL https://arxiv.org/abs/2305.18290.

[23] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. In *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, 2017. URL https://arxiv.org/abs/1707.06347.

[24] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

[25] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS 2023)*, New Orleans, LA, USA, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

[26] Yu Wang, Chi Han, Tongtong Wu, Xiaoxin He, Wangchunshu Zhou, Nafis Sadeq, Xiusi Chen, Zexue He, Wei Wang, Gholamreza Haffari, Heng Ji, and Julian McAuley. Towards lifespan cognitive systems, 2025. URL https://arxiv.org/abs/2409.13265.

[27] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 52040–52094. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5d413e48f84dc61244b6be550f1cd8f5-Paper-Datasets_and_Benchmarks_Track.pdf.

[28] Ran Xu, Yuchen Zhuang, Yue Yu, Haoyu Wang, Wenqi Shi, and Carl Yang. Rag in the wild: On the (in)effectiveness of llms with mixture-of-knowledge retrieval augmentation. *arXiv preprint arXiv:2507.20059*, 2025.

[29] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

[30] Wujiang Xu, Kai Mei, Hang Gao, Juntao Tan, Zujie Liang, and Yongfeng Zhang. A-mem: Agentic memory for llm agents, 2025. URL `https://arxiv.org/abs/2502.12110`.

[31] Hui Yang, Sifu Yue, and Yunzhong He. Auto-gpt for online decision making: Benchmarks and additional opinions. *arXiv preprint arXiv:2306.02224*, 2023. doi: 10.48550/arXiv.2306.02224. URL `https://doi.org/10.48550/arXiv.2306.02224`.

[32] 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.

[33] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *Proceedings of the 11th International Conference on Learning Representations (ICLR 2023)*, 2023. URL `https://openreview.net/forum?id=WE_vluYUL-X`.

[34] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *NeurIPS (Workshops)*, 2024. State-of-the-art agents (e.g. GPT-4o) succeed on <50

[35] Yuyan Zhou, Liang Song, Bingning Wang, and Weipeng Chen. Metagpt: Merging large language models using model exclusive task arithmetic. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1711–1724, Miami, Florida, USA, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.102.