EvoMem: Improving Multi-Agent Planning with Dual-Evolving Memory

Wenzhe Fan [1,†] **Ning Yan** [2] **Masood Mortazavi** [2]

[1] University of Illinois Chicago
[2] Futurewei Technologies

Abstract

Planning has been a cornerstone of artificial intelligence for solving complex problems, and recent progress in LLM-based multi-agent frameworks have begun to extend this capability. However, the role of human-like memory within these frameworks remains largely unexplored. Understanding how agents coordinate through memory is critical for natural language planning, where iterative reasoning, constraint tracking, and error correction drive the success. Inspired by working memory model in cognitive psychology, we present EvoMem, a multi-agent framework built on a dual-evolving memory mechanism. The framework consists of three agents (Constraint Extractor, Verifier, and Actor) and two memory modules: Constraint Memory (CMem), which evolves across queries by storing task-specific rules and constraints while remains fixed within a query, and Query-feedback Memory (QMem), which evolves within a query by accumulating feedback across iterations for solution refinement. Both memory modules are reset at the end of each query session. Evaluations on trip planning, meeting planning, and calendar scheduling show consistent performance improvements, highlighting the effectiveness of EvoMem. This success underscores the importance of memory in enhancing multi-agent planning.

1 Introduction

Planning is a fundamental cognitive ability that involves generating sequences of actions and reasoning about future states [11, 28]. However, prior work has shown that large language models (LLMs) often struggle with planning tasks, especially those requiring multi-step reasoning or long-range dependencies [12, 26, 22]. At the same time, the crucial role of memory in supporting computational reasoning, planning, and other cognitive functions has been widely recognized across computing, computational neuroscience, and psychology [21, 19, 2, 10, 5]. Yet, while memory has been understood as useful to reasoning and adaptation, its structure and mechanisms in the natural language planning tasks remain largely underexplored in LLM-based multi-agent frameworks.

Inspired by working memory model in cognitive psychology [3], we propose EvoMem, a multi-agent framework that explicitly incorporates the dual-evolving memory into the planning process. The framework assembles three agents (Constraint Extractor, Verifier, and Actor) and two memory modules. Given a query, the Constraint Extractor identifies task-specific rules and constraints, establishing the Constraint Memory (CMem). This fixed memory anchors the Actor's solution generation across iterations and updates only when a new query is introduced. The Verifier evaluates the Actor's output and provides feedback for subsequent iterations, forming the Query-feedback

[†]Work done during the author's internship at Futurewei Technologies. Correspondence to: wfan23@uic.edu

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Bridging Language, Agent, and World Models for Reasoning and Planning.

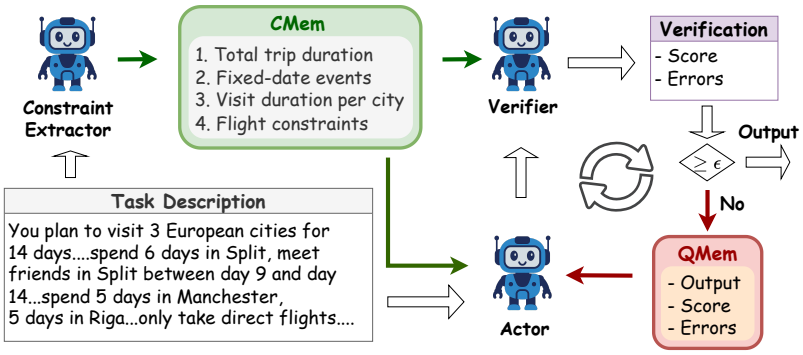


Figure 1: The dual-evolving memory mechanism in EvoMem: CMem evolves across queries to establish fixed, task-specific constraints for the actor and verifier. QMem evolves within a single query by recording failed attempts (solution, score, errors), providing iterative feedback until the solution is verified or the turn limit is reached.

Memory (QMem). This dynamic memory evolves within a single query, enabling solution refinement throughout the ongoing task. CMem and QMem are reset after each query.

EvoMem is conceptually aligned with Baddeley's multi-component theory of working memory first proposed in 1974 [3] (Details shown in App. B). CMem plays a role analogous to the phonological loop by maintaining all constraints in a stable, verbalized form across rounds. QMem functions similarly to the sketchpad, accumulating intermediate outputs, verifier feedback, and error signals into a coherent multi-round representation. The iterative Actor–Verifier process serves as a central executive that selectively attends to CMem, updates QMem, and directs the next reasoning step. This cognitive framework explains why combining long-lived constraints with dynamically updated, per-round information leads to a more consistent and effective planning process.

We evaluate EvoMem on the NaturalPlan benchmark, covering trip planning, calendar scheduling, and meeting planning tasks. Using Gemini-1.5-Pro [24] as the primary backbone, EvoMem outperforms strong baselines, achieving average gains of +11.17% in trip planning, +2.56% in calendar scheduling, and +3.76% in meeting planning. To assess model-agnostic performance, we further test on DeepSeek V3 [15] and GPT-4.1-mini [1], and observe consistent improvements across models.

In summary, our contributions are as follows: First, we present EvoMem, a multi-agent framework that integrates specialized agent roles with dual-evolving memory mechanism to address complex natural language planning tasks. Second, we demonstrate that EvoMem achieves state-of-the-art performance on the NaturalPlan benchmark, validating the effectiveness of its memory-centric design. Finally, we show that for complex planning tasks, maintaining per-query memory is sufficient to capture iterative reasoning and substantially improve performance.

2 EvoMem

We propose EvoMem, a multi-agent framework for solving complex planning tasks through an iterative self-correction process. As illustrated in Figure 1, the framework employs two complementary evolving memory modules, i.e., CMem and QMem, to guide reasoning and ensure that solutions progressively align with all specified constraints.

2.1 Multi-Agent Design

Constraint Extractor Constraints are the rules and limitations that a valid plan must satisfy. For instance, in trip planning tasks, constraints may involve the total trip duration, fixed-date events (e.g., weddings or conferences), city-specific stays, and available flights. Prior work [20] indicates the criticality of constraint identification for reliable plan verification. The constraint extractor derives task-specific constraints from the problem description, capturing the essential conditions that any candidate plan must satisfy. These constraints are then appended to the actor's prompts, ensuring its reasoning remains grounded in the core requirements of the task.

Verifier The verifier evaluates each solution against the extracted constraints and facilitates refinement through QMem-mediated feedback. This agent provides two types of output: (i) *Reward Score (0-100)* that measures plan quality based on constraint satisfaction, and (ii) *Feedback* (including constraint violations) which the agent deposits in QMem for future iterations. To ensure full compliance, only plans with a perfect score of 100 are accepted.

Actor The actor generates a solution for a given query. The actor takes the query and the extracted constraints (CMem), incorporating feedback from QMem in subsequent turns if an earlier turn fails. Prompts and examples for these three agents are given in App. D and App. E.

2.2 Memory Modules

Given a query, our framework generates a solution through the iterative process of up to T turns. Planning is jointly guided by the Constraint Memory (CMem), which adapts across queries but remains fixed within a single query, and the Query-feedback Memory (QMem), which evolves across iterations within the same query. Together, these two modules form what we call "a dual-evolving memory" mechanism. CMem and QMem are reset at the end of each query session.

Constraint Memory (CMem) CMem stores the constraints extracted from the query and is appended to the actor's prompt at every turn. It remains fixed throughout the multi-turn refinement process, ensuring that the actor consistently adheres to the core requirements of the problem. The importance of CMem is further validated in our ablation study (Sec. 3.2).

Query-feedback Memory (QMem) QMem is a dynamic memory module that evolves across iterations by recording information from failed attempts in earlier turns. Whenever a candidate solution violates constraints, its solution, score, and identified errors are logged in QMem. This memory evolves across at most T iterations, guiding subsequent refinements.

2.3 Proposed Framework

The planning process for a given query unfolds over T iterations, with each turn progressively refining the solution under the guidance of memory.

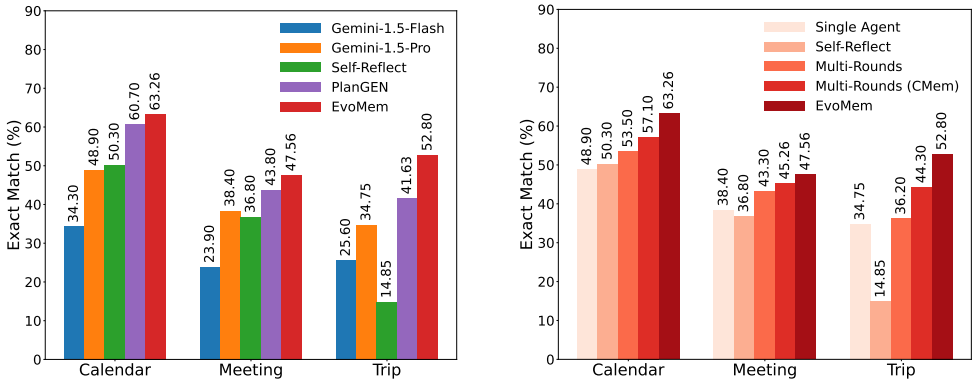
Constraint Extraction: First, the constraint extractor identifies all relevant constraints from the query and records them in CMem. This module remains fixed throughout the query session, serving as a stable anchor for all subsequent iterations. ***Solution Generation***: The actor then generates an initial solution based solely on the constraints in CMem. In later iterations, solution generation is enhanced by also incorporating the accumulated feedback stored in QMem. ***Verification***: Next, the verifier evaluates the candidate solution against the stored constraints in CMem, producing a reward score and listing any violated constraints. If no violations are found, the solution is accepted. ***Self-Correction***: If the verifier finds violations, the framework enters the Self-Correction loop. The failed solution, its score, and the specific errors are all logged as a new entry in QMem. In subsequent iterations, the actor leverages both the fixed constraints in CMem and the accumulated feedback in QMem to refine its output. The cycle of generation, verification, and memory update continues until either a valid solution is found or the maximum number of T turns is reached. ***Reset***: Finally, both CMem and QMem are cleared before a new query begins, preventing any cross-contamination.

3 Experiment

3.1 Main Result

We conduct experiments on three datasets in NaturalPlan [35]: 1,000 instances each for Calendar Scheduling and Meeting Planning, and 1,600 instances for Trip Planning. For comparison, we adopt three baselines: (i) Zero-shot CoT [13], (ii) Self-Reflect [20], where the same model iteratively refines outputs through self-reflective feedback loops, and (iii) PlanGen [20], a strong method that achieves state-of-the-art results in multi-agent frameworks across these datasets. In EvoMem, we set the temperature of the constraint extractor to 0.1, the verifier to 0, and the actor to 0.7. We set the maximum number of iterations T to 5.

As shown in Figure 2a, EvoMem achieves the highest exact match scores [35] across all tasks averaged over 5 runs: 63.26$\pm$0.41% (Calendar), 47.56$\pm$0.32% (Meeting), and 52.08$\pm$0.12% (Trip). All agents in these experiments are powered by Gemini-1.5-Pro. These results highlight the effectiveness of EvoMem in diverse natural language planning tasks and establish a new state-of-the-art among



(a) Performance (b) Effect of different components

Figure 2: (a) Performance comparison of EvoMem against other baseline methods. (b) Performance comparison of EvoMem gradually adding one more component. (All experiments are conducted with Gemini-1.5-Pro.)

multi-agent framework methods. They further confirm the crucial role of query-specific memory in advancing the planning capabilities of LLM-based multi-agent systems.

Methods	Trip	Calendar	Meeting	Methods	Trip	Calendar	Meeting	Methods	Trip	Calendar	Meeting
Gemini-1.5-Pro	34.75%	48.9%	38.4%	DeepSeek v3	38.5%	57%	46.3%	GPT-4.1-mini	24.75 %	45.6%	37.2%
EvoMem (Gemini-1.5-Pro)	53.5%	63.26%	47.56%	EvoMem (DeepSeek v3)	49%	61.1%	50.2%	EvoMem (GPT-4.1-mini)	39.1%	66.8%	33.71%

Table 1: Performance comparison of EvoMem with three base LLM models on three different planning tasks.

3.2 Ablation Study

Effect of Different Components Figure 2b presents the ablation study evaluating the contribution of each component in our framework. We begin with a "Single Agent" setting, where only the actor is used. Adding "Self-Reflect" enables multi-round self-evaluation but produces mixed results, emphasizing the necessity of an explicit verifier guided by rules. Incorporating both the constraint extractor and verifier in the "Multi-Rounds" yields consistent improvements, underscoring the importance of constraint-based verification. Performance is further enhanced by introducing the CMem module, which ensures that the actor remains aligned with the query's constraints across iterations. Finally, adding QMem, which accumulates errors from previous attempts, provides additional gains, demonstrating the benefit of evolving, query-feedback memory in iterative planning.

Effect of Different LLMs The results in Table 1 demonstrate the impact of EvoMem across different LLM backbones. With Gemini-1.5-Pro, EvoMem achieves the strongest overall performance, improving Trip by +18.75%, Calendar by +14.36%, and Meeting by +9.16% relative to the base model. Comparable gains are observed with DeepSeek v3, where EvoMem adds +10.5% on Trip, +4.1% on Calendar, and +3.9% on Meeting. Even with the smaller GPT-4.1-mini, EvoMem substantially enhances performance, particularly on Calendar (+21.2%), though Meeting shows a slight decline (–3.5%). While absolute performance varies across different backbones, EvoMem consistently delivers relative improvements, indicating that its memory-driven mechanism generalizes well across models.

Effect of Self-Correction Rounds We observe that the performance remains stable across maximum iteration number $T \in \{3, 5, 7\}$, with an average of $53.284\% \pm 0.357\%$. A cost-benefit analysis on the iteration cap K (with $T = 7$) revealed diminishing returns, as each additional iteration yields fewer new successes. The first three iterations ($K = 3$) are highly effective, successfully resolving 93.7% of all solvable tasks while using only 67.88% of the total queries. Extending the cap from $K = 3$ to $K = 5$ yields 45 additional successes (from 93.7% to 97.6%) for only 2.8% more dataset coverage. Going from $K = 5$ to $K = 7$ nets just 28 more successes to complete the dataset, the most expensive part for the smallest gain. These findings indicate that $T = 5$ as the optimal balance point of performance and efficiency, while $T = 3$ is a decent choice for cost-sensitive applications. The details of this study are shown in App. C.

Effect of Temperature We first fix the temperatures of verifier and constraint extractor while varying the actor's temperature from 0.7 to 0.5 and 0.3. We then evaluate two additional settings: all agents with temperature 0 (fully deterministic) and all agents with temperature 1 (Gemini default). The results show a variance of only 0.256%, indicating that the framework is highly robust to temperature settings.

4 Conclusion and Future work

In this work, we introduced EvoMem, a dual-evolving memory framework for natural language planning. EvoMem integrates three agents (Constraint Extractor, Actor, and Verifier) with two evolving memory modules: a Constraint Memory (CMem) that evolves at the query level to set fixed, task-specific constraints for a given task, and a Query-feedback Memory (QMem) that evolves at the iteration level within a query by accumulating multi-turn feedback. Experiments on trip planning, meeting planning, and calendar scheduling show consistent gains across diverse LLM backbones, demonstrating the value of EvoMemo and further emphasizing the importance of memory in advancing multi-agent planning. The updated working memory theory incorporates long-term memory, suggesting promising directions for extending EvoMem toward richer long-term or learned workflow memories to support more complex planning and reasoning tasks.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Kristijan Armeni, Marko Pranjić, and Senja Pollak. Transformer verbatim in-context retrieval across time and scale. *Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

[3] Alan Baddeley. Working memory: Theories, models, and controversies. *Annual review of psychology*, 63(1):1–29, 2012.

[4] Bernd Bohnet, Azade Nova, Aaron T Parisi, Kevin Swersky, Katayoon Goshvadi, Hanjun Dai, Dale Schuurmans, Noah Fiedel, and Hanie Sedghi. Exploring and benchmarking the planning capabilities of large language models. *arXiv preprint arXiv:2406.13094*, 2024.

[5] Thackery I Brown, Stephanie A Gagnon, and Anthony D Wagner. Stress disrupts human hippocampal-prefrontal function during prospective spatial navigation and hinders flexible behavior. *Current Biology*, 30(10):1821–1833, 2020.

[6] Jiefeng Chen, Jie Ren, Xinyun Chen, Chengrun Yang, Ruoxi Sun, Jinsung Yoon, and Sercan Ö Arık. Sets: Leveraging self-verification and self-correction for improved test-time scaling. *arXiv preprint arXiv:2501.19306*, 2025.

[7] Weizhe Chen, Sven Koenig, and Bistra Dilkina. Reprompt: Planning by automatic prompt engineering for large language models agents. *arXiv preprint arXiv:2406.11132*, 2024.

[8] 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.

[9] Andy Clark. *Supersizing the Mind: Embodiment, Action, and Cognitive Extension*. Oxford University Press, 12 2008. ISBN 9780195333213. doi: 10.1093/acprof:oso/9780195333213.001.0001. URL `https://doi.org/10.1093/acprof:oso/9780195333213.001.0001`.

[10] Peter M Gollwitzer and Paschal Sheeran. Psychology of planning. *Annual review of psychology*, 76(1):303–328, 2025.

[11] Fangkai Jiao, Chengwei Qin, Zhengyuan Liu, Nancy F Chen, and Shafiq Joty. Learning planning-based reasoning by trajectories collection and process reward synthesizing. *arXiv preprint arXiv:2402.00658*, 2024.

[12] Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. Llms can't plan, but can help planning in llm-modulo frameworks. *arXiv preprint arXiv:2402.01817*, 2024.

[13] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

[14] Kuang-Huei Lee, Ian Fischer, Yueh-Hua Wu, Dave Marwood, Shumeet Baluja, Dale Schuurmans, and Xinyun Chen. Evolving deeper llm thinking. *arXiv preprint arXiv:2501.09891*, 2025.

[15] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[16] Lei Liu, Xiaoyan Yang, Yue Shen, Binbin Hu, Zhiqiang Zhang, Jinjie Gu, and Guannan Zhang. Think-in-memory: Recalling and post-thinking enable llms with long-term memory. *arXiv preprint arXiv:2311.08719*, 2023.

[17] Yanming Liu, Xinyue Peng, Jiannan Cao, Shi Bo, Yuwei Zhang, Xuhong Zhang, Sheng Cheng, Xun Wang, Jianwei Yin, and Tianyu Du. Tool-planner: Task planning with clusters across multiple tools. *arXiv preprint arXiv:2406.03807*, 2024.

[18] Kai Mei, Xi Zhu, Wujiang Xu, Wenyue Hua, Mingyu Jin, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. Aios: Llm agent operating system. *arXiv preprint arXiv:2403.16971*, 2024.

[19] Ida Momennejad. Memory, space, and planning: Multiscale predictive representations. *arXiv preprint arXiv:2401.09491*, 2024.

[20] Mihir Parmar, Xin Liu, Palash Goyal, Yanfei Chen, Long Le, Swaroop Mishra, Hossein Mobahi, Jindong Gu, Zifeng Wang, Hootan Nakhost, et al. Plangen: A multi-agent framework for generating planning and reasoning trajectories for complex problem solving. *arXiv preprint arXiv:2502.16111*, 2025.

[21] Dale Schuurmans. Memory augmented large language models are computationally universal. *arXiv preprint arXiv:2301.04589*, 2023.

[22] Parshin Shojaee, Iman Mirzadeh, Keivan Alizadeh, Maxwell Horton, Samy Bengio, and Mehrdad Farajtabar. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity, 2025. URL `https://arxiv.org/abs/2506.06941`.

[23] Yimin Tang, Yurong Xu, Ning Yan, and Masood Mortazavi. Enhancing long context performance in llms through inner loop query mechanism. *NeurIPS (Workshop on Adaptive Foundation Models)*, 2024.

[24] Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

[25] Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Advances in Neural Information Processing Systems*, 36: 38975–38987, 2023.

[26] Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models-a critical investigation. *Advances in Neural Information Processing Systems*, 36:75993–76005, 2023.

[27] Bing Wang, Xinnian Liang, Jian Yang, Hui Huang, Shuangzhi Wu, Peihao Wu, Lu Lu, Zejun Ma, and Zhoujun Li. Enhancing large language model with self-controlled memory framework. *arXiv preprint arXiv:2304.13343*, 2023.

[28] Evan Wang, Federico Cassano, Catherine Wu, Yunfeng Bai, Will Song, Vaskar Nath, Ziwen Han, Sean Hendryx, Summer Yue, and Hugh Zhang. Planning in natural language improves llm search for code generation. *arXiv preprint arXiv:2409.03733*, 2024.

[29] Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P Xing, and Zhiting Hu. Promptagent: Strategic planning with language models enables expert-level prompt optimization. *arXiv preprint arXiv:2310.16427*, 2023.

[30] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

[31] Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. Os-copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*, 2024.

[32] Chengxing Xie and Difan Zou. A human-like reasoning framework for multi-phases planning task with large language models. *arXiv preprint arXiv:2405.18208*, 2024.

[33] Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*, 2024.

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

[35] Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V Le, Ed H Chi, et al. Natural plan: Benchmarking llms on natural language planning. *arXiv preprint arXiv:2406.04520*, 2024.

[36] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19724–19731, 2024.

[37] Yuqi Zhu, Shuofei Qiao, Yixin Ou, Shumin Deng, Shiwei Lyu, Yue Shen, Lei Liang, Jinjie Gu, Huajun Chen, and Ningyu Zhang. Knowagent: Knowledge-augmented planning for llm-based agents. *arXiv preprint arXiv:2403.03101*, 2024.

A Related Work

LLM Agents for Planning Recent research has established that Large Language Models (LLMs) struggle to solve planning tasks directly from task descriptions [12, 22, 26]. This limitation has spurred the development of various natural language planning benchmarks and environments to study the problem comprehensively [35, 4, 33, 25]. In this work, we evaluate our approach on NaturalPlan [35], which includes trip planning, calendar scheduling, and meeting planning tasks.

In response to this challenge, a significant body of work has focused on creating LLM-based agent frameworks to enhance planning capabilities. Some approaches are tailored to specific domains [17, 31], Other methods [37, 32] focus on integrating external information to guide the planning process. Another line of research concentrates on optimizing the prompts themselves [7, 29].

The current SOTA LLM-based agent framework on natural language planning benchmarks is held by PlanGen [20]. While recent works continue to make progress, they have primarily focused on innovations within single-agent frameworks [14, 6].

Our work contributes to this field by investigating a different axis of improvement: the role of the memory mechanism. We demonstrate that by incorporating memory into a simple multi-agent framework, our approach achieves better result than PlanGen on NaturalPlan benchmark.

Memory for LLM Agents The importance of memory in computational, planning and other cognitive tasks has been well established, e.g., by computing, computational neuroscience, cognition and psychology researchers [21, 19, 2, 9, 10, 5]. Memory mechanisms have become central to advancing LLM-based agents, as effective planning and reasoning often require retaining and reusing information beyond a single context window. Early efforts focused on memory utilization [8, 31, 18, 36, 16, 27], using short-term and long-term memory to support information storage and recall. More recent work has explored agentic memory [34, 23], where agents dynamically organize memories, establish new connections, and evolve their knowledge with experience. Another important direction is workflow memory [30], which captures and reuses common routines to guide future tasks and improve efficiency. However, none of the above methods investigate the effectiveness of memory mechanisms in natural language planning tasks. Our framework specifically addresses this gap.

B Working Memory Model in Cognitive Psychology

Figure 3 shows the classic working memory model with three parts: the central executive, the visuo-spatial sketchpad, and the phonological loop.

The central executive is like the “manager" of your mind’s working memory. It controls your attention, decides what matters, and coordinates the other components. The visuo-spatial sketchpad is the “inner eye". It holds visual and spatial information—like imagining a map, remembering where objects are, or picturing a diagram—and becomes active whenever you visualize something in your head. The phonological loop is the “inner voice”. It keeps words or sounds in your mind for a short time. For example, you repeat a phone number to yourself so you do not forget it.

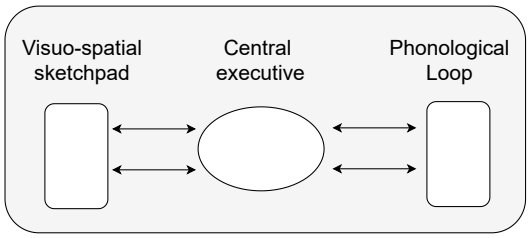


Figure 3: Working memory model proposed in 1974 [3].

C Ablation: Effect of Self-Correct Rounds

iteration cap K	#queries finished $\leq$ K	% of dataset	#Success (score=100) $\leq$ K	% of all successes	+ Successes vs prev. cap	Success remaining
3	1,086	67.88%	1,086	93.7%	-	73 (6.3%)
4	1,111	69.44%	1,111	95.86%	+25	48 (4.14%)
5	1,131	70.69%	1,131	97.58%	+20	28 (2.42%)
6	1,146	71.63%	1,146	98.88%	+15	13 (1.12%)
7	1,600	100%	1,159	100%	+13	0 (0%)

Table 2: Success coverage (score=100) vs. iteration gap K (maximum iteration number $T = 7$).

D Details on LLM Agents

Constraint extractor Prompt

You are an expert constraint extraction agent for planning problems. Your task is to extract only structured constraints in a clean, concise, and strictly formatted style."

Extract constraints using only the format below: {Output Format}

(Optional) There is information might include:
{You may provide the specific constraints you want}
{You may provide example query and corresponding extracted constraints}
Query:{query}

Verifier Prompt

You are a meticulous verifier responsible for evaluating trip plans against a set of hard constraints. You must assess both the numeric durations and logical feasibility of the plan. Be strict and precise.
(Optional) {Verification steps}

Given query, Please evaluate whether the plan satisfies all the constraints.
Query:
{query}

Input plan:
{solution}

Constraints:
{constraints}

Format your response strictly as follows - no extra text, comments, or explanations:
Score: [integer number, reward score between 0 and 100]
Violated Constraints: [string, list any constraints that the plan violates or any errors in the plan, and provide the reason why each constraint was violated]

Actor Prompt

You are an expert at planning problems. {dataset specific description}.
Along with the task, you are also given 5 example tasks and solutions. You can learn from the examples to generate your own plan on the given task.
Follow the format of the example solution and the required output format precisely, starting with {dataset specific description}.
Only provide the travel plan, no other text.

Query:
{query}

Constraints:
Your solution must strictly satisfy the following constraints.
{constraints}
If any constraint is violated, the solution is considered incorrect

Previous Errors:
The following notes lists why previous attempts failed:
{errors}
Do not repeat the same errors.

Based on all the information above, generate your final output.

E Examples of Different Agents in EvoMem

Constraint Extractor (Trip Planning)

Problem:

You plan to visit 3 European cities for 14 days in total.
You only take direct flights to commute between cities.
You would like to visit Florence for 6 days.
You want to meet a friend in Florence between day 9 and day 14.
You would like to visit Barcelona for 5 days.
You would like to visit Helsinki for 5 days.
Here are the cities that have direct flights: Barcelona and Florence; Helsinki and Barcelona.

Generated Constraints:

1. Total trip duration: 14 days
2. Visit durations per city:
   – Florence: 6 days (Day 9–14)
   – Barcelona: 5 days
   – Helsinki: 5 days
3. Fixed-date events: meet friend in Florence between Day 9–14
4. Flight constraints:
   – Direct flights only
   – Direct flight connections: Barcelona–Florence, Helsinki–Barcelona

Table 3: Example input and extracted constraints for the constraint extractor in trip planning.

Constraint Extractor (Calendar Scheduling)

Problem:

You need to schedule a meeting for Michelle, Steven, and Jerry for one hour between the work hours of 9:00 to 17:00 on Monday.

Here are the existing schedules for everyone during the day:

- Michelle has meetings on Monday during 11:00–12:00;
- Steven has blocked their calendar on Monday during 9:00–9:30, 11:30–12:00, 13:30–14:00, 15:30–16:00;
- Jerry has blocked their calendar on Monday during 9:00–9:30, 10:00–11:00, 11:30–12:30, 13:00–14:30, 15:30–16:00, 16:30–17:00.

Generated Constraints:

1. Meeting Participants: Michelle, Steven, Jerry
2. Meeting Duration: 1 hour (60 minutes)
3. Meeting Day: Monday
4. Allowed Meeting Hours: 09:00–17:00
5. Blocked Schedules:
   – Michelle: 11:00–12:00
   – Steven: 09:00–09:30, 11:30–12:00, 13:30–14:00, 15:30–16:00
   – Jerry: 09:00–09:30, 10:00–11:00, 11:30–12:30, 13:00–14:30, 15:30–16:00, 16:30–17:00
6. Meeting Preferences: None

Table 4: Example input and extracted constraints for the constraint extractor in calendar scheduling.

Constraint Extractor (Meeting Planning)

Problem:

You are visiting San Francisco for the day and want to meet as many friends as possible.

Solve the problem by considering various schedules and picking the best one to optimize your goals.

Travel distances (in minutes):

- Marina District → Mission District: 20
- Mission District → Marina District: 19

You arrive at Marina District at 9:00AM.

Stephanie will be at Mission District from 10:30AM to 1:30PM.

You'd like to meet Stephanie for a minimum of 120 minutes.

Generated Constraints:

1. Arrival Time Constraint:
   – You arrive at Marina District at 9:00 AM
2. Travel Time Constraints:
   – Marina District → Mission District: 20 minutes
   – Mission District → Marina District: 19 minutes
3. Availability Constraint:
   – Stephanie is at Mission District from 10:30 AM–1:30 PM
4. Meeting Duration Constraint:
   – You want to meet Stephanie for at least 120 minutes
5. Optimization Goal:
   – Maximize the number of friends met during the day by considering various schedules.

Table 5: Example input and extracted constraints for the constraint extractor in meeting planning.

Verifier (Trip Planning)

Constraints:

1. Total trip duration: 12 days
2. Visit durations per city:
   – Tallinn: 2 days (Day 11–12)
   – Seville: 7 days
   – Munich: 5 days
3. Timing constraints: meet friend in Tallinn between Day 11–12
4. Flight constraints:
   – Direct flights only
   – Direct flight connections: Seville–Munich, Munich–Tallinn

Solution:

Here is the trip plan for visiting the 3 European cities for 12 days:
Day 1–7: Arrive in Seville and visit for 7 days.
Day 7: Fly from Seville to Munich.
Day 7–10: Visit Munich for 5 days.
Day 10: Fly from Munich to Tallinn.
Day 10–12: Visit Tallinn for 2 days.

Score: 90
Errors: Munich visit duration is incorrect: the plan states 5 days but the itinerary covers 4 days (Day 7–10).

Table 6: Verifier example for trip planning.

Verifier (Calendar Scheduling)

Constraints:

1. Meeting Participants: Arthur, Michael, and Samantha
2. Meeting Duration: 1 hour (60 minutes)
3. Meeting Day: Monday
4. Allowed Meeting Hours: Between 09:00 and 17:00
5. Blocked Schedules:
   – Arthur: 09:00–09:30, 10:30–12:00, 16:00–17:00
   – Michael: 13:00–13:30, 14:00–14:30
   – Samantha: 10:30–11:00, 12:00–15:00, 15:30–17:00
6. Meeting Preferences: None
7. Feasibility: True (explicitly stated "there exists a solution")

Solution:

Proposed meeting time: Monday, 12:00–13:00

Score: 0
Errors: Samantha is busy from 12:00 to 15:00, overlapping with the proposed meeting time of 12:00–13:00.

Table 7: Verifier example for calendar scheduling

Verifier (Meeting Planning)

Constraints:

1. Arrival Time Constraint:
   – You arrive at Chinatown at 9:00 AM
2. Travel Time Constraints:
   – Chinatown → Russian Hill: 7 minutes
   – Russian Hill → Chinatown: 9 minutes
3. Availability Constraint:
   – Ronald is at Russian Hill from 3:15 PM to 9:30 PM
4. Meeting Duration Constraint:
   – You want to meet Ronald for at least 105 minutes
5. Optimization Goal:
   – Maximize the number of friends met during the day by considering various schedules

Solution:

1. Arrival Time Constraint:
   – You arrive at Chinatown at 9:00 AM
2. Travel Time Constraints:
   – Chinatown → Russian Hill: 7 minutes
   – Russian Hill → Chinatown: 9 minutes
3. Availability Constraint:
   – Ronald is at Russian Hill from 3:15 PM to 9:30 PM
4. Meeting Duration Constraint:
   – You want to meet Ronald for at least 105 minutes
5. Optimization Goal:
   – Maximize the number of friends met during the day by considering various schedules

Score: 90

Errors: Meeting Duration Constraint: The meeting with Ronald only lasts 45 minutes (from 3:15 PM to 4:00 PM), which is less than the required minimum duration of 105 minutes.

Table 8: Verifier example for meeting planning: constraints, proposed solution, score, and detected errors.