代理驅動的滲透測試工作記憶預過濾框架

謝明軒[1]、郭振忠[2]
[1]南臺科技大學資訊工程系、[2]南臺科技大學資訊工程系
[1]MB3G0103@stust.edu.tw、[2]spotkuo@stust.edu.tw

論文摘要

大型語言模型（LLM）驅動的自主代理在網路安全領域展現潛力，但長序列推理容易因對話累積造成上下文溢出與資訊遺失，進而影響推理品質。本研究提出一種工作記憶管理機制：在工具輸出寫入記憶前，先由分析者代理進行蒸餾與結構化摘要，僅保留與當前攻擊階段相關的關鍵資訊，以控制上下文規模並維持資訊完整性。實驗以 CTF 網頁安全題目為場景，在統一的 ReAct [1] 推理框架下，與三種記憶策略比較（ReAct[1]、HiAgent[2]、CHAP[3]），所有方法共用相同提示、工具與模型。並於 picoCTF 題目上重複測試，以奪旗成功率、工具呼叫次數、token 消耗量作為指標。結果顯示，所提方法於簡單題目成功率達 100%，平均 token 消耗降低 13%~36%；困難題目成功率為 40%（其餘 0%~20%），token 消耗降低 6%~50%。

關鍵詞：大型語言模型、資訊安全、網站滲透測試、記憶機制

前言

隨著 GPT-4 等大型語言模型展現出強大的推理和程式碼生成能力，研究者開始將它們與工具呼叫機制結合，打造能夠自主和環境互動的智慧代理。Wang 等人 [4] 對 LLM 自主代理進行了系統性整理，指出記憶模組是代理架構的核心組成之一，直接影響代理在長序列任務中的推理穩定性。Zhang 等人 [5] 進一步針對記憶機制進行分類與分析，說明現有方法在長序列任務中面臨的上下文管理挑戰。這類代理通常基於 ReAct（Reasoning + Acting）[1] 框架運作，先推理再行動，並根據觀察結果動態調整策略，已經在程式開發、資料分析等領域展現潛力。

CTF（Capture The Flag）是一種常見的資訊安全競賽形式，參與者須透過分析系統或應用程式中的漏洞來取得隱藏的旗標（flag）。其中，網頁安全題目是評估此類代理能力的理想場景。PentestGPT [6] 是將 LLM 應用於自動化滲透測試的代表性工作，其研究發現 LLM 在個別子任務上表現良好，但在維持整體測試上下文方面仍存在困難，進一步說明了記憶管理在滲透測試任務中的重要性。解題過程需要進行多輪偵查、假設建立與漏洞利用，通常涉及多次工具呼叫。然而，隨著推理輪次增加，對話歷史持續累積，導致上下文規模不斷膨脹，逐漸超出 LLM 可有效處理的範圍。這不僅會使早期關鍵資訊被截斷，也會降低模型的推理穩定性，出現重複嘗試失敗路徑或偏離原始提示詞（prompt drift）的情況，甚至因超出 token 限制而中斷任務。

針對上述問題，現有研究提出了不同的記憶管理策略，像是 HiAgent [2] 的子目標壓縮、CHAP [3] 的上下文接力等。但這些方法大多使用各自的實驗設定，缺少在統一條件下的直接比較，很難判斷記憶機制本身對代理表現的影響。

因此，本研究提出一種工作記憶（Working Memory）搭配分析者代理的方法，由獨立的 LLM 在每次工具呼叫後蒸餾出結構化摘要，讓主代理始終在可控的上下文規模中運作。

為了確保比較公平，本研究在相同的 ReAct 推理框架下比較四種記憶策略，共用相同的推理框架、工具集和基礎模型，只有記憶管理機制不同，以 picoCTF 題目進行多次實驗，從成功率、token 消耗和工具呼叫次數等方面評估各方法的差異。

文獻探討

ReAct（Reasoning and Acting）[1] 是一種用於大型語言模型（LLM）的框架，它將推理（reasoning）和行動（acting）以交錯方式結合，使模型在執行任務時能同時產生內部推理軌跡與具體動作。具體而言，ReAct 透過交互生成「思考（Thought）」與「行動（Action）」序列，讓模型能根據觀察結果調整行動計畫，並進一步收集外部資訊以支持後續推理與決策。此方法在問答、事實驗證及互動決策任務中，相較於僅依賴推理生成答案的模型，能提供更高的成功率並生成更具體可解釋的策略軌跡，是目前多數 LLM agent 系統的基本架構。

然而，ReAct 本身並未針對長序列上下文膨脹提出特定策略。其方法為所有歷史對話與觀察都作為上下文輸入，但在複雜任務中，對話歷史會快速累積，導致上下文膨脹。這會讓模型推理變得不穩定，例如：重複嘗試已經失敗的路徑、偏離提示詞，甚至可能因超過 token 限制而中斷任務。

HiAgent [2] 提出了一種基於子目標的工作記憶管理策略，用於長序列決策任務。它將任務動態拆解成子目標（subgoal），並在每個子目標完成後，將子目標的動作與觀察序列摘要成簡短記憶，只保留與當前階段相關的重要資訊；必要時，還可檢索完整歷史以支持後續決策。HiAgent 本身並非為滲透測試或 CTF 設計，本研究僅借鑑其記憶管理策略，將子目標驅動的上下文壓縮應用於長序列

攻擊任務，並評估其控制上下文膨脹的效果。

CHAP（Context Relay for Long-Running Penetration-Testing Agents）[3] 是針對長序列滲透測試任務設計的框架，提出了上下文接力（Context Relay）機制。把歷史對話與操作序列分段，並在代理完成一段任務後將摘要傳遞給下一階段，維持可控的上下文規模，避免模型因長序列膨脹而出現推理不穩定。這種分段接力策略有助於長任務的穩定性，同時保留關鍵操作的歷史摘要。

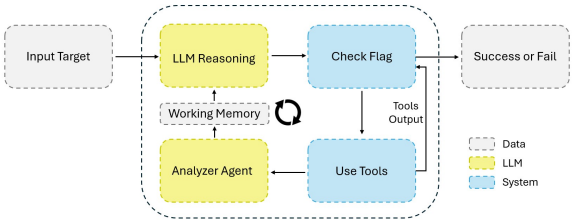
Reflexion [7] 提出了另一種記憶管理思路，讓代理在每次任務嘗試結束後，以自然語言反思失敗原因，並將反思內容寫入記憶緩衝區，作為下一次嘗試的額外上下文。這種方式屬於寫入後反思，與本研究在工具輸出寫入記憶前進行蒸餾的方向不同。此外，Reflexion 依賴多輪重複嘗試來累積學習，較不適用於每次執行獨立的 CTF 單次解題場景。

在上下文壓縮技術方面，LongLLMLingua [8] 提出針對長文本的 Prompt 壓縮方法，透過移除對當前推理貢獻較低的 token 來縮減輸入規模。然而此類方法主要針對靜態文件或單步推理場景設計，對於需要持續累積工具輸出的動態長序列代理任務，壓縮過程可能損失後續推理所需的關鍵資訊。本研究採用結構化摘要而非 token 級壓縮，以保留攻擊狀態的完整語意。

研究方法

一、系統架構設計

本研究提出一種結合工作記憶（Working Memory）與分析者代理（Analyzer Agent）的 LLM 自主代理架構，主要是為了解決長序列推理過程中因對話累積導致的上下文溢出問題。整體架構以 ReAct 框架為基礎，由主代理負責推理與行動，並在每次工具呼叫後引入獨立的分析者代理進行資訊蒸餾，如圖一所示。



圖一、系統架構圖

主代理的推理流程如下：主代理依據 ReAct 框架進行思考並決定工具呼叫。在工具執行前後，系統會先行偵測主代理的思考內容或工具輸出中是否已包含 flag；如果沒有找到 flag，則將工具輸出結果傳遞給分析者代理進行摘要蒸餾，再將摘要寫入工作記憶。主代理於下一輪推理時，僅讀取工作記憶內容與上一輪對話作為上下文，而非完整的歷史對話，藉此將上下文規模維持在可控範圍內。

分析者代理接收工具的原始輸出後，依照固定的六欄位結構生成摘要，各欄位定義與範例如表一所示：（1）已發現端點（Discovered Endpoints）、（2）已知資料（Known Data）、（3）Flag 狀態（Flag Status）、（4）已嘗試的狀態（Attempted States）、（5）未嘗試的組合（Untested Combinations）、（6）目前假設（Current Hypotheses）。此結構化設計確保摘要僅保留與當前攻擊階段直接相關的資訊，同時提供主代理明確的下一步推理依據。

表一、工作記憶欄位定義與範例

欄位名稱	說明	範例
已發現端點	攻擊者已探測到的主機或服務端點列表	https://xxx.picoctf.net/ stylex.css Index.js
已知資料	已取得的目標資訊	Server: nginx/1.19.7 Content Type: text/html Form method: POST with fields `username`, `password`
Flag 狀態	列出 Flag	Not Found Found:picoCTF{xxxxx}
已嘗試的狀態	已經嘗試過的攻擊組合或策略	Access css/xdfgwd.html | 404 Not Found (2 attempts)
未嘗試的組合	尚未嘗試的攻擊組合，供後續決策用	1. Analyze SpongeBob_SquarePants.gif for hidden data. 2. Explore myess.css for any additional clues.
目前假設	攻擊者對目標環境或漏洞的暫時假設	The primary vulnerability was the insecure logic in `index.js` revealing the flag through encoded credentials.

本研究所有代理均統一採用 GPT-4o 作為基礎語言模型，主代理可呼叫 Kali Linux 環境內的完整工具集，包含但不限於 curl、gobuster、sqlmap 等網頁滲透測試常用工具。

二、比較基準方法

為確保實驗比較的公平性，本研究將所提方法與三種記憶策略進行對比，所有方法共用相同的推理框架（ReAct）、工具集（Kali Tools）與基礎模型（GPT-4o），僅記憶管理機制不同：

Working Memory：本研究所提之方法，每次工具執行後，由分析者代理將工具輸出蒸餾為六欄位結構化摘要，主代理只讀取最新摘要與上一輪對話繼續推理。

ReAct：基於 ReAct 原論文記憶策略，不進行任何記憶壓縮，完整保留所有歷史對話作為上下文。

CHAP：基於 CHAP 原論文程式碼改寫而來，採用上下文接力策略，在每輪推理時以滾動方式傳遞精簡後的歷史摘要。

HiAgent：基於 HiAgent 原論文程式碼改寫而來，採用子目標壓縮策略，將對話歷史依據任務子目標進行分段摘要。

其中 HiAgent 與 CHAP 由本研究依各自論文重新實作，並與 ReAct 和 Working Memory 統一整合至相同的推理框架與實驗流程中，確保比較條件一致。

三、實驗設計

(一) 實驗環境

本研究實驗環境採用 Kali Linux 2025.4，所有工具皆使用系統內建工具集，並未另行安裝額外套件。所有方法共用相同的執行環境，確保實驗條件一致。

(二) 題目選擇

本研究選擇了 picoCTF 平台作為驗證記憶策略的基準，使用了兩道難度不同的網頁安全題目進行實驗，分別為簡單題目與複雜題目，難度分類依據人工解題所需的工具呼叫次數來定義，如表二所示。

簡單題目：題目為 picoCTF login，雖然 picoCTF 標示該題目為中等難度，但人工解題實際上只需三步以內：首先檢視網路請求即可找到 index.js，讀取後即可看到 base64 編碼的 flag，因此歸類為簡單題目。

複雜題目：題目為 picoCTF Secrets，同樣標示為中等難度，但人工解題時經過了多輪偵查與假設驗證，工具呼叫次數遠超過三步，因此歸類為複雜題目。

表二、實驗題目難度定義

難度	定義	題目	漏洞類型
簡單	三步以內完成（工具呼叫次數 ≤ 3）	picoCTF login	原始碼洩漏
複雜	超過三步才能完成（工具呼叫次數 > 3）	picoCTF Secrets	存取權限控管失效 (Broken Access Control,BAC)

(三) 提示詞設計

本研究針對主代理與分析者代理分別設計不同的提示詞。

分析者代理的提示詞要求模型以固定的六欄位結構輸出摘要。當出現重複失敗的嘗試時，代理會將相同的錯誤結果合併紀錄，避免重複佔用上下文空間；對於新加入的未嘗試組合，則會在前方標記[NEW]，方便主代理快速識別尚未嘗試的攻擊方向。

主代理的提示詞以網頁 CTF 解題為核心，將解題流程劃分為四個階段：觀察（Observe）、地圖建立與假設（Map & Hypothesize）、測試與利用（Test & Exploit）、以及旗標擷取（Flag Extraction）。在觀察階段，主代理被要求先理解目標應用程式的技術架構，包含伺服器類型、框架、輸入點等，而非直接進行目錄掃描。進入假設階段後，提示詞提供了針對不同技術的攻擊向量對應關係，例如 PHP 對應 SQLi 與 LFI、Flask 對應 SSTI、Express 對應 NoSQL 注入等，引導代理根據觀察結果選擇最可能的攻擊方向。

提示詞中也內建了決策規則，例如當同一攻擊向量失敗兩次後應切換策略、遇到 403 時嘗試路徑繞過、遇到 500 時讀取錯誤訊息等，避免代理陷入無效的重複嘗試。此外，代理被要求優先參考工作記憶中標有 [NEW] 的未嘗試組合，確保每一輪推理都在探索新的攻擊方向，縮短解題所需的推理輪次。

(四) 最大輪次限制

為避免代理無限制地消耗 token，本研究統一將所有方法的最大推理輪次設定為 25 輪，達到上限後強制停止並視為失敗。此設定對所有方法一致，不影響比較的公平性。

(五) 實驗次數

每道題目對每種方法各執行五次，避免單次結果偶然造成偏差，最後以平均值作為比較依據。

(六) 評估指標

本研究從三個角度評估各方法的表現，如表三所示：

奪旗成功率：記錄五次執行中成功找到 flag 的次數，計算方式為成功次數除以總執行次數（5 次），以百分比表示。

推理輪次：記錄每次執行所使用的推理輪數（上限 25 輪），用以觀察各方法的解題過程長度。

Token 消耗量：統計每次完整執行所消耗的總 token 數，直接反映各記憶管理機制對上下文規模的控制效果。

表三、評估指標定義

指標	說明	計算方式
奪旗成功率	代理成功找到flag的比率	成功次數 / 執行次數 * 100%
推理輪次	代理執行的總輪數(最大25輪)	紀錄每次執行的輪數取平均
Token消耗量	每次完整執行所消耗的總token數	紀錄每次執行的token數取平均

研究結果

本研究以 picoCTF 平台上兩道難度不同的網頁安全題目進行實驗，各方法執行五次後取平均，結果分別如表四（簡單題目）與表五（複雜題目）所示。

表四、picoCTF Login（簡單題目）實驗結果

方法	推理輪次	平均 Token 消耗	成功率
Working Memory（本研究）	**2.8**	**15,445**	100%
ReAct	4.6	17,849	100%
CHAP	5.6	24,080	100%
HiAgent	8.8	23,773	100%

表五、picoCTF Secrets（複雜題目）實驗結果

方法	推理輪次	平均 Token 消耗	成功率
Working Memory（本研究）	23.8	**136,470**	**40%**
ReAct	25	270,681	0%
CHAP	**23.4**	157,455	20%
HiAgent	24.4	145,738	20%

從數據中可以看出，在簡單題目中，四種方法均達到 100% 的奪旗成功率。然而在 token 消耗與推理效率上，本研究所提之工作記憶方法表現最

佳，平均 token 消耗為 15,445，相較於 ReAct、CHAP、HiAgent 分別減少約 13%、36%、35%；平均推理輪次僅 2.8 輪，為四種方法中最低，顯示分析者代理的結構化摘要能有效引導主代理快速定位關鍵資訊，避免重複嘗試。

而在複雜題目中，各方法的差異更為明顯。本研究方法成功率達到 40%，為四種方法中最高。ReAct 完全未能解題(0%)，CHAP 與 HiAgent 各達 20%。在 token 消耗方面，本研究方法平均消耗 136,470 tokens，相較於 ReAct 減少約 50%，相較於 CHAP 與 HiAgent 分別減少約 13% 與 6%。值得注意的是，ReAct 在困難題中五次執行均達到 25 輪上限仍未解題，顯示未經記憶管理的方法在長序列任務中容易陷入重複推理或上下文溢出，嚴重影響解題能力。CHAP 與 HiAgent 雖有部分成功，但 token 消耗仍明顯高於本研究方法，且推理輪次接近上限，反映其記憶壓縮效果有限。

結論

本研究提出一種結合工作記憶與分析者代理的記憶管理機制，在每次工具呼叫後由分析者代理將輸出蒸餾成六欄位結構化摘要，讓主代理在可控的上下文規模下持續推理。實驗結果顯示，在簡單題上四種方法成功率相同，但本研究方法的 token 消耗與推理輪次均為最低；在困難題上本研究方法成功率達 40%，明顯優於其他三種方法，同時 token 消耗也是最少的。

整體來看，預先蒸餾的設計確實能在資訊寫入工作記憶前就完成篩選，避免不相關的工具輸出佔用上下文，在長序列的困難題中這個效果特別突出。相較之下，ReAct 在困難題中五次全部失敗，說明沒有記憶管理機制的代理在面對複雜任務時有明顯的上限。

不過值得注意的是，本研究方法 token 消耗較低，有一部分原因來自成功率較高。成功拿到 flag 的時候通常在較少輪次內就結束，自然消耗較少 token。從實驗數據也可以觀察到，每一輪引入分析者代理進行摘要蒸餾本身就需要額外的 token 開銷，因此在所有方法同樣失敗的情況下，本研究方法的 token 消耗不一定會比其他方法低。這點在困難題的數據中也有所體現本研究方法與 HiAgent 的 token 消耗差距（約 6%）相較於簡單題的差距（約 35%）明顯縮小，反映出當任務複雜度提高、執行輪次拉長時，分析者代理的額外開銷會逐漸顯現。

未來研究方向包括以下幾個面向：首先，擴大實驗題目的數量與難度範圍，納入更多類型的網頁安全漏洞（如 SQL injection、SSRF 等），以驗證本方法在不同攻擊情境下的穩定性。其次，進一步探討分析者代理的摘要品質對主代理推理表現的影響，例如評估不同摘要結構對資訊保留率的差異。此外，也可考慮讓分析者代理根據當前攻擊階段動態調整摘要欄位，而非固定使用六欄位結構，以提升對不同類型題目的適應性。

參考文獻

[1] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in *Proc. 11th Int. Conf. Learning Representations (ICLR)*, 2023. [Online]. Available: https://openreview.net/forum?id=WE_vluYUL-X

[2] M. Hu, T. Chen, Q. Chen, Y. Mu, W. Shao, and P. Luo, "HiAgent: Hierarchical working memory management for solving long-horizon agent tasks with large language model," in *Proc. 63rd Annu. Meeting Assoc. Comput. Linguistics (ACL)*, Vol. 1: Long Papers, 2025, pp. 32779–32798. [Online]. Available: https://aclanthology.org/2025.acl-long.1575/

[3] M. Vangeli, J. Brynielsson, M. Cohen, and F. Kamrani, "Context relay for long-running penetration-testing agents," in *Proc. Workshop LLM Assisted Security and Trust Exploration (LAST-X)*, NDSS Symp., 2026. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/lastx2026-42.pdf

[4] L. Wang et al., "A survey on large language model based autonomous agents," *Front. Comput. Sci.*, vol. 18, 2024. [Online]. Available: https://link.springer.com/article/10.1007/s11704-024-40231-1

[5] Z. Zhang et al., "A survey on the memory mechanism of large language model based agents," *ACM Trans. Inf. Syst.*, 2024. [Online]. Available: https://dl.acm.org/doi/10.1145/3748302

[6] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, "PentestGPT: Evaluating and harnessing large language models for automated penetration testing," in *Proc. 33rd USENIX Security Symp.*, Philadelphia, PA, 2024, pp. 847–864. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/deng

[7] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," in *Adv. Neural Inf. Process. Syst.*, vol. 36 (NeurIPS 2023), 2023. [Online]. Available: https://openreview.net/forum?id=vAElhFcKW6

[8] H. Jiang, Q. Wu, X. Luo, D. Li, C.-Y. Lin, Y. Yang, and L. Qiu, "LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression," in *Proc. 62nd Annu. Meeting Assoc. Comput. Linguistics (ACL)*, Bangkok, Thailand, 2024, pp. 1658–1677. [Online]. Available: https://aclanthology.org/2024.acl-long.91/