Pre-Act: Multi-Step Planning and Reasoning Improves Acting in LLM Agents

Mrinal Rawat[1], **Ambuje Gupta**[1], **Rushil Goomer**[1], **Alessandro Di Bari**[1], **Neha Gupta**[1], **Roberto Pieraccini**[1]
[1]Uniphore
Correspondence: rawatmrinal06@gmail.com

Abstract

The ReAct (Reasoning + Action) capability in large language models (LLMs) has become the foundation of modern agentic systems. Recent LLMs, such as DeepSeek-R1 and OpenAI o1/o3, exemplify this by emphasizing reasoning through the generation of ample intermediate tokens, which help build a strong premise before producing the final output tokens. In this paper, we introduce Pre-Act, a novel approach that enhances the agent's performance by creating a multi-step execution plan along with the detailed reasoning for the given user input. This plan incrementally incorporates previous steps and tool outputs, refining itself after each step execution until the final response is obtained. Our approach is applicable to both conversational and non-conversational agents. To measure the performance of task-oriented agents comprehensively, we propose a two-level evaluation framework: (1) turn level and (2) end-to-end. Our turn-level evaluation, averaged across five models, shows that our approach, Pre-Act, outperforms ReAct by 70% in Action Recall on the Almita dataset. While this approach is effective for larger models, smaller models crucial for practical applications, where latency and cost are key constraints, often struggle with complex reasoning tasks required for agentic systems. To address this limitation, we fine-tune relatively small models such as Llama 3.1 (8B & 70B) using the proposed Pre-Act approach. Our experiments show that the fine-tuned 70B model outperforms GPT-4, achieving a 69.5% improvement in action accuracy (turn-level) and a 28% improvement in goal completion rate (end-to-end) on the Almita (out-of-domain) dataset.

1 Introduction

Management of dialog systems has been a key focus of research in human-machine interaction. Various models have emerged, broadly categorized into distinct approaches (Pieraccini and Huerta, 2005). Although inference and reinforcement-learning-based methods (Levin and Pieraccini, 1997) have been explored extensively, commercial applications have favored pragmatic solutions. Traditional rules-based or call-flow-driven systems (Lieberman, 1997) rely on predefined state transitions and procedural definitions. Although robust, they impose rigid dialog structures and require extensive manual design, leading to prolonged development cycles and high deployment costs, limiting scalability.

With the emergence of LLM-driven agents, dialog systems have evolved beyond static rule-based flows. LLMs enable open-ended interactions where business users are not constrained by predefined prompts, fostering more flexible, goal-oriented conversations. Agentic AI (Huang et al., 2024) acts autonomously, making decisions using advanced machine learning. However, effectiveness hinges on the underlying orchestrator, typically an LLM requiring strong reasoning and function-calling abilities. Recent models like DeepSeek-R1 (DeepSeek-AI et al., 2025) and OpenAI o1/o3 generate additional reasoning-focused tokens and demonstrate these abilities, which is why approaches such as ReAct (Yao et al., 2023b) have become the primary foundation upon which today's agentic frameworks are built.

While ReAct provides a strong foundation for building agents, previous implementations had limitations. The reasoning component (generated as "thought") typically focuses only on the reasoning required for the immediate action, making it inadequate for handling complex tasks that require executing a sequence of actions. Although the LLM is the system's cognitive backbone that drives reasoning, decision-making, and tool use, agent performance declines if the LLM orchestrator lacks intelligence or contextual awareness. Unfortunately, advanced reasoning remains largely restricted to proprietary models such as GPT-4 and Claude, lim-

iting broader adoption.

To address these limitations, we make the following contributions in this paper:

- Pre-Act, an enhanced version of ReAct that improves agents performance by generating a multi-step plan along with detailed reasoning for each action for the given task. Steps are executed sequentially, incorporating past actions and observations as context, refining the plan until the final output is obtained.
- A fine-tuning strategy leveraging datasets adapted for Pre-Act, enabling smaller models (e.g., Llama 8B, 70B) to match or surpass proprietary LLMs up to 20 times larger while reducing latency and cost, a critical aspect for real-world agentic applications.
- A two-level evaluation: (a) turn-level, assessing whether predicted actions align with ground truth and (b) end-to-end, measuring goal completion and progress rate.

2 Related Work

The concept of autonomous agents and agentic architectures is not new. The Open Agent Architecture (Martin et al., 1999), developed at SRI in the 1990s, enabled service creation through distributed autonomous agents coordinated by one or more facilitators. The Galaxy Architecture (Seneff et al., 1998), initially developed at MIT for spoken dialogue systems, became the reference for the DARPA Communicator program, using a central HUB to exchange messages with multiple specialized servers.

Recent work has explored various strategies to enhance LLM reasoning and execution through advanced prompting techniques. Chain-of-Thought (CoT) prompting (Wei et al., 2022b) enables LLMs to break down complex problems into intermediate steps, while Self-Consistency (Wang et al., 2023) improves reliability by generating multiple reasoning paths. ReAct (Yao et al., 2023b) integrates reasoning with action execution, further extended by Tree-of-Thought (ToT) (Yao et al., 2023a) and Graph-of-Thought (Besta et al., 2024), which introduce structured reasoning paths. Other approaches, such as Least-to-Most prompting (Zhou et al., 2023) and Chain-of-Verification (Dhuliawala et al., 2023), enhance efficiency and reliability. However, these methods often generate redundant tokens and struggle with coherence across multiple steps, particularly in smaller models where computational efficiency is critical.

Fine-tuning is emerging as an effective method for achieving strong performance while using fewer tokens in the prompt (Ye et al., 2025). Techniques such as instruction tuning (Wei et al., 2022a) and RLHF (Ouyang et al., 2022), when applied to smaller models with limited, but high-quality data, have shown promise in enhancing reasoning capabilities. However, these methods generally require datasets with explicit planning mechanisms to effectively learn structured reasoning and function-calling abilities, which are not readily available to the public and are difficult to obtain.

In this work, we address these limitations by introducing structured plans along with detailed reasoning generated by LLM. While prior work has explored planning in LLMs (Fu et al., 2023), our approach extends it by generating a detailed multi-step plan with reasoning for each step, applicable to both conversational and non-conversational AI agents. Additionally, existing evaluation frameworks for LLM agents often assess either individual action accuracy (Arcadinho et al., 2024) or overall task performance (Gioacchini et al., 2024), but not both. Our two-level evaluation framework provides a more comprehensive assessment, considering both granular actions and overall task success.

3 Approach

3.1 Incremental Multi-step planning

While the ReAct approach has emerged as a fundamental paradigm for building LLM-based agents, its current implementation typically focuses on single-step reasoning and (immediate) action generation (Yao et al., 2023b). For each input request, the agent iteratively performs up to n tool calls $[0, n]$ before arriving at a final answer. Despite its effectiveness, this approach often struggles with complex tasks that require long-term planning and sequential decision-making. To address this limitation, we propose Pre-Act, which generates a comprehensive multi-step plan. Instead of generating isolated reasoning steps, our approach formulates a structured execution plan with the reasoning that not only integrates previously executed steps, but also outlines the forthcoming steps necessary to achieve the final goal.

For a given input request, our approach generates a plan consisting of $n + 1$ sequential steps

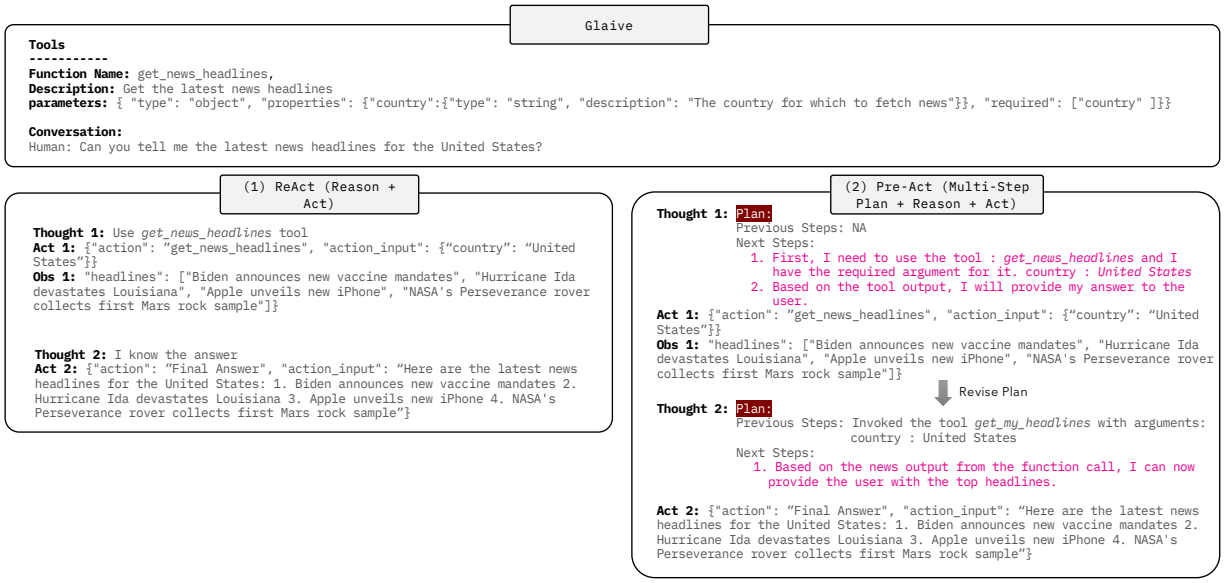

Figure 1: Side-by-side comparison of ReAct and Pre-Act (our approach) for an example use-case from the Glaive dataset. While this example demonstrates a single tool call, the approach can be extended to multiple tool calls.

$S\ \{s_1, s_2, \ldots, s_n, s_{fa}\}$, where each step s_i explicitly specifies the intended action with a detailed thought, and the last step s_{fa} represents the final answer. When a step s_i involves a tool call, the corresponding action a_i is executed, and the resulting observation o_i is incorporated into the reasoning process. For the subsequent step s_{i+1}, the system leverages both the current observation o_i and the accumulated context $C_t = \{(a_1, o_1), \ldots (a_i, o_i)\}$, to refine the plan and generate the next action, enhancing decision-making efficiency (see Figure 1 for an example). This refinement process is particularly valuable when the outcome of a previous step deviates from expectations or results in failure, as it allows the agent to adapt its strategy dynamically. This iterative incorporation of past actions and observations augments the context C_t, ensuring that the agent maintains coherence and adaptability across multi-step interactions. For the input prompt template, please refer to Appendix D.

3.2 Curriculum learning : Agents

To train the models for agents, we adopt a curriculum learning (Bengio et al., 2009) approach through incremental fine-tuning. Our process consists of two key stages:

- **Initial Fine-tuning:** We first fine-tune the pre-trained Llama models on the Glaive dataset using the ReAct approach with minimal reasoning. This decision was pragmatic as the dataset is extensive and adding comprehensive reasoning annotations (as required for Pre-Act) would be very expensive. At this stage, our primary goal is to enable the model to perform agentic tasks specifically, learning to distinguish when to make tool calls with appropriate parameters versus when and what to generate as a response.

- **Progressive Refinement:** Using the trained checkpoint from the first stage, we further fine-tune the model on our proprietary dataset using the Pre-Act approach while preserving learning from the previous step through LoRA training, which modifies only a small fraction of parameters (Wistuba et al., 2023). For this smaller and high-quality dataset, we leverage our organization's annotation team to generate reasoning for each step. Importantly, the sequence of steps (multi-step plan) leading to the final answer was derived from the dataset itself; only the reasoning for each step was provided by annotators.

3.2.1 Dataset for Fine-Tuning

Training models for conversational agentic capabilities requires datasets with two critical components: multi-turn dialogues and function (tool) calls with their corresponding responses. Such datasets are rare, and existing ones are not directly compatible with Pre-Act training requirements. The following section details our methodology for transforming the two datasets we used to train the models, ensuring alignment with our format requirements.

Glaive Dataset We primarily leveraged the Glaive Function-Calling v2 dataset[1], which covers conversational use-cases across multiple domains. Each data point comprises system instructions, tool definitions, and chat history. The chat history consists of user-assistant interactions, where assistant responses may incorporate function calls (FC) along with their corresponding function responses (FR). Formally, each conversation turn follows the structure: {[USER],[(FC_1, FR_1). . . (FC_n, FR_n)] ,[ASSISTANT]} where FC_i represents a function call and FR_i denotes its response, (FC_n, FR_n) are optional depending on the specific interaction context. Algorithm 1 describes the transformation of each conversation turn into appropriate input-output pairs based on whether they contain tool calls or not. The sample output for both cases can be found in Appendix B.

Proprietary Dataset While the Glaive dataset provides a foundation for basic agentic capabilities, it has several limitations: minimal conversational deviations, lack of exception handling, and at most one consecutive tool call. To train more robust models, we curated a proprietary dataset spanning over 100 use-cases across domains including healthcare, manufacturing, telecommunications, banking, and finance. This dataset introduces complex scenarios that are absent in Glaive. Although the raw dataset also follows the same turn-based format, we specifically adapted it for Pre-Act using the methodology outlined in Section 3.1. Unlike Glaive, where we provide minimal reasoning (extractable from the dataset) for the tool calls and final answer, Pre-Act requires explicit reasoning at each step. Extending Algorithm 1, we structured the dataset so that at each turn, the subsequent steps (i.e., tool calls) leading to the final answer could be derived. We incorporated placeholders for reasoning at each step, which were then completed by expert annotators. These annotators provided detailed explanations for every decision, ensuring a clear rationale behind tool calls and final responses (example in Fig. 3).

Algorithm 1: Pseudo-code for transforming the dataset to follow ReAct

Input: Conversation Turns $T = \{\text{USER}, [(FC_1, FR_1), \ldots, (FC_n, FR_n)], \text{ASSISTANT}\}$, Instruction I, Tools Definition TD

$D \leftarrow \emptyset$; // Dataset
foreach $t_i \in T$ **do**
  if $FC \notin t_i$ **then**
    $Ip_i \leftarrow \{I, TD, t_i(\text{USER})\}$
    $Op_i \leftarrow$ Thought: I know the final answer. Action : t_i(ASSISTANT) // ASSISTANT response
    $D \leftarrow D \cup \{(Ip_i, Op_i)\}$
  else
    $D_i \leftarrow \emptyset$;
    for $j \in \{1, \ldots, n\}$ **do**
      // N function calls
      $Ip_j \leftarrow \{I, TD, t_i(\text{USER}), C((a_1, o_1), \ldots, (a_{j-1}, o_{j-1}))\}$ // C is the accumulated context till $j-1$
      $Op_j \leftarrow$ Thought: Need to invoke tool : FC_j
      Action : FC_j(arguments)
      $D_i \leftarrow D_i \cup \{(Ip_j, Op_j)\}$
    $Ip_i \leftarrow \{I, TD, t_i(\text{USER}), C\}$
    $Op_i \leftarrow$ Thought: I know the final answer. Action : t_i(ASSISTANT)
    $D_i \leftarrow D_i \cup \{(Ip_i, Op_i)\}$
    $D \leftarrow D \cup D_i$

4 Evaluation

4.1 Dataset

We conducted our experiments on three datasets: Glaive, our proprietary dataset, and Almita. For Almita, we apply the same transformation as described in the previous section (annotated by our team for Pre-Act). The statistics of these datasets relevant for Level-1 evaluation are presented in Table 1. For Level 2 evaluation (end-to-end testing), we use the Almita dataset. Unlike Level 1, this stage requires running actual conversations with tool calling, needing an environment with tool implementations. Hence, we conduct our end-to-end evaluation on five complex Almita use-cases, illustrated in Table 3. We selected these from 18 available cases by manually filtering out similar ones and those lacking tools or workflow information. Our contributions to this dataset, along with results and simulated conversations, are available in our Github repository.[2]

[1] glaive-function-calling-v2

[2] https://github.com/acl2025-submission/acl2025

Dataset	# Train	# Val	# Test	#Use-cases	#Tools
Glaive	341200	2672	4021	6702	1060
Proprietary	1852	128	378	119	360
Almita	-	-	1100	18	70

Table 1: Statistics of the three datasets used in Level-1 evaluation describing the number of instances used in training, validation, and test sets, respectively, along with total use-cases and tools.

4.2 Level 1: Turn Level Evaluation

At this level, we evaluate each conversation turn for correctness against the ground truth (g.t.). For each given request, the action prediction from the LLM is either a Final Answer or a tool call. From a metrics perspective, we first calculate action recall. If the g.t. action is a tool call, we measure the F1 score and parameter match (full). If it is a Final Answer, we compute the F1 score and use a similarity model (Xiao et al., 2023) to assess similarity to the g.t..

4.3 Level 2: End-to-End (E2E) Evaluation

A key limitation of the Level-1 evaluation framework is that it focuses solely on individual turns rather than evaluating the conversation or task as a whole. Most publicly available datasets consist primarily of "happy path" scenarios, where users provide correct and straightforward information. However, real-world interactions are often more unpredictable, where users may deviate from topics, providing partial or incorrect information or errors introduced through automatic speech recognition (ASR) in voice interfaces. To address such scenarios and gain a comprehensive, end-to-end view of AI agent performance, we introduce the Level-2 evaluation that assesses an AI agent's ability to manage complex conversations within a business context, where the primary goal is to accomplish a predefined task following a particular workflow.

4.3.1 Aspects of Level 2 Evaluation

Milestone Creation: We draw inspiration from AgentQuest (Gioacchini et al., 2024), which highlights the importance of breaking down tasks into mission-critical milestones and introduces metrics like progress rate and goal completion to measure an agent's task completion performance. However, in our setting, defining milestones and evaluating progress is not straightforward, as tasks can vary significantly. We utilize GPT-4 to create a structured set of incremental and/or conditional milestones that align with the user-defined workflow and tools (refer to Appendix F for the prompt). Additionally, we prompt it to establish dependencies between these milestones. The resulting output is then used to construct a milestone dependency graph. Since this automatically generated graph may not always be entirely accurate, we incorporate human verification and refinement to ensure its correctness. Once finalized, this graph is stored for that use-case and serves as an input for agent evaluation (sample provided in Appendix G).

Simulating Environments: An effective approach to end-to-end evaluation of conversational systems is using a synthetic user to simulate interactions based on predefined scenarios, a concept introduced in Levin et al. (2000) and later expanded in Pietquin and Dutoit (2006). To test our AI agents, we built a simulation environment leveraging GPT-4 as a synthetic user and let it interact with the AI agent. By dynamically assigning personas, we model user behaviors ranging from simple queries to complex ethical dilemmas, security challenges, and critical decision-making. This ensures rigorous assessment of the AI's adherence to business workflows and handling of sensitive situations, enhancing its trustworthiness.

Evaluating Goal Completion Metrics: We use LLM-as-a-judge (Zheng et al., 2023) (GPT4) to evaluate simulated conversations. For goal completion, the LLM considers the entire conversation, the AI agent's task (instructions), available tools, and then reviews stored milestones to verify whether they were achieved successfully in the correct sequence. To calculate the progress rate, we map the achieved milestones (predicted by the LLM) against the milestone dependency graph and measure progress by traversing from the start node through the completed milestones, calculating the ratio of the distance covered to the total distance from start to end. This provides a measure of how far the conversation has progressed toward the final goal. If 100% of the milestones are completed to reach the end, the goal is considered fully achieved (prompt given in Appendix H).

5 Results and Discussion

Table 2 highlights two key findings: (a) the impact of Pre-Act on pretrained models (first five in the table) compared to ReAct, and (b) a comparison of fine-tuned models using Pre-Act against others. Pre-Act consistently outperforms ReAct across action recall, tool calls, and final answer similarity on

Models	Approach	In-Domain Test Dataset										Out-of-Domain Dataset				
		Glaive					Proprietary					Almita				
		Tool			Final Answer		Tool			Final Answer		Tool			Final Answer	
		Action Recall	F1	Params Match (Full)	F1	Sim.	Action Recall	F1	Params Match (Full)	F1	Sim.	Action Recall	F1	Params Match (Full)	F1	Sim.
llama3.1 8B (van.)	ReAct	0.6064	0.5447	0.7868	0.6532	0.8586	0.1528	0.1290	0.1556	0.1903	**0.9413**	0.1537	0.1906	0.2477	0.1101	0.6277
	Pre-Act	-	-	-	-	-	**0.4613**	**0.5087**	**0.4967**	**0.3697**	0.9324	**0.2779**	**0.3702**	**0.7216**	**0.1293**	**0.7239**
llama3.1 70B (van.)	ReAct	0.7771	0.767	0.7995	0.7859	0.8971	0.3791	0.4214	0.5269	0.3314	**0.9395**	0.3268	0.3643	0.5677	0.2724	0.7853
	Pre-Act	-	-	-	-	-	**0.5472**	**0.5609**	**0.5677**	**0.5246**	0.9137	**0.4045**	**0.4204**	**0.7508**	**0.3846**	**0.8842**
Nvidia Nemotron 70B(van.)	ReAct	0.6771	0.6050	0.6578	0.7478	0.8878	0.2377	0.1988	0.2258	0.3002	0.8013	0.1460	0.0647	0.1259	0.2659	0.7673
	Pre-Act	-	-	-	-	-	**0.4928**	**0.4966**	**0.4774**	**0.4864**	**0.8955**	**0.3908**	**0.4049**	**0.7443**	**0.3728**	**0.8370**
Deepseek-distil llama-3.1-70B(van.)	ReAct	0.7823	0.7611	0.7314	0.8004	0.9040	0.2756	0.2339	0.2168	0.3555	0.6965	0.3232	0.3725	0.6586	0.2511	**0.8438**
	Pre-Act	-	-	-	-	-	**0.6531**	**0.6712**	**0.4421**	**0.6435**	**0.7157**	**0.5036**	**0.4697**	**0.7518**	**0.5488**	0.8418
gpt-4-turbo	ReAct	0.9265	0.8649	0.9145	0.9496	0.9449	0.4933	0.5150	0.5612	0.5057	0.8870	0.4430	0.3995	0.6634	0.4930	0.7452
	Pre-Act	-	-	-	-	-	**0.6131**	**0.6019**	**0.5870**	**0.6293**	**0.9116**	**0.5449**	**0.4616**	**0.7532**	**0.6214**	**0.8201**
llama3.1-8B (f.t.)	Pre-Act	0.9881	0.9744	0.9289	0.9922	0.9623	0.8911	0.8327	0.5806	0.9363	0.8894	0.8706	0.7464	0.7475	0.9306	0.8256
llama3.1-70B (f.t.)	Pre-Act	**0.9929**	**0.9848**	**0.9586**	**0.9954**	**0.9826**	**0.9111**	**0.8625**	**0.6258**	**0.9523**	0.8671	**0.9238**	**0.8636**	**0.7961**	**0.9496**	**0.8861**

Table 2: Results on two in-domain test sets (Glaive and a proprietary dataset) and one out-of-domain set (Almita). "Van." refers to the vanilla (pretrained) model, while "f.t." denotes fine-tuned. Higher values indicate better performance. Bold numbers highlight the better approach between ReAct and Pre-Act, while bold and underlined numbers indicate the best overall model in each column.

Models	Approach	Use-cases									
		Order Discrepancy		Internet Ping		Gift Card		Digital Download		Delivery	
		GC	PR	GC	PR	GC	PR	GC	PR	GC	PR
gpt-4-turbo	ReAct	0.28	0.39	0.00	0.11	0.18	0.33	0.55	0.79	0.60	0.74
	Pre-Act	0.63	0.72	0.52	0.73	0.75	0.74	0.68	0.90	**0.66**	**0.89**
llama-3.1 70B(f.t.)	Pre-Act	**0.75**	**0.89**	**1.00**	**1.00**	**0.90**	**0.86**	**0.89**	**0.97**	0.60	0.79

Table 3: End-to-End Evaluation results on Almita Dataset use-cases. GC represents Goal Completion and PR represents progress rate.

both the proprietary and Almita datasets, with a minor drop in final answer similarity for Llama 3.1 8B & 70B on the proprietary dataset. On average (not shown in the table), Pre-Act improves action recall over ReAct by **102%** on the proprietary dataset and **70%** on Almita. A comparison with Glaive was not possible due to missing Pre-Act annotations.

The second part of our results highlights the impact of fine-tuning. On the in-domain test sets, our fine-tuned 70B model outperforms GPT-4 and its vanilla counterpart by 7.1% and 27.7% on the Glaive dataset. To assess generalization, we also evaluated on the Almita dataset (out-of-domain—applicable in case of f.t.), where our fine-tuned 70B model outperformed GPT-4 (Pre-Act) by 69.5% and its vanilla counterpart by 128% in action recall. The consistent performance gains both in-domain and out-of-domain underscore the robustness of our fine-tuning approach. Our findings highlight that smaller fine-tuned models can match or even surpass the performance of significantly larger proprietary systems, making them an efficient and scalable alternative.

Table 3 presents the end-to-end results for conversations generated synthetically. We ran the simulation 50 times across all five use-cases and reported the average goal completion (GC) rate and progress rate. As shown in Table 3, our fine-tuned model, despite being significantly smaller in size, outperforms both the GPT with ReAct and Pre-Act approaches in most use-cases. On average (not shown in the table), our model achieves a GC rate of **0.82**, compared to **0.32** for GPT with ReAct and **0.64** for GPT with Pre-Act. It is noteworthy that the Pre-Act approach with GPT demonstrates significant improvements compared to GPT with ReAct, while our fine-tuned model achieves even higher GC and progress rates in challenging scenarios.

6 Conclusion and Future Work

We have introduced Pre-Act, a method that improves LLM agent performance through multi-step planning and reasoning. We also proposed a fine-tuning strategy that allows smaller models to achieve performance comparable to larger LLMs with lower latency and cost, along with a two-level evaluation framework for rigorous assessment. In the future, we aim to enhance model robustness by incorporating complex scenarios and recovery paths into training data and adopting more deterministic evaluation methods to mitigate the volatility in LLM-as-a-judge assessments.

References

Samuel Arcadinho, David Aparicio, and Mariana S. C. Almeida. 2024. Automated test generation to evaluate tool-augmented llms as conversational ai agents. *Preprint*, arXiv:2409.15934.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA. Association for Computing Machinery.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michał Podstawski, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2024. Graph of Thoughts: Solving Elaborate Problems with Large Language Models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. 2023. Chain-of-verification reduces hallucination in large language models. *Preprint*, arXiv:2309.11495.

Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. 2023. Complexity-based prompting for multi-step reasoning. *Preprint*, arXiv:2210.00720.

Luca Gioacchini, Giuseppe Siracusano, Davide Sanvito, Kiril Gashteovski, David Friede, Roberto Bifulco, and Carolin Lawrence. 2024. Agentquest: A modular benchmark framework to measure progress and improve llm agents. *Preprint*, arXiv:2404.06411.

Qiuyuan Huang, Naoki Wake, Bidipta Sarkar, Zane Durante, Ran Gong, Rohan Taori, Yusuke Noda, Demetri Terzopoulos, Noboru Kuno, Ade Famoti, Ashley J. Llorens, John Langford, Hoi Vo, Fei-Fei Li, Katsushi Ikeuchi, and Jianfeng Gao. 2024. Agent ai towards a holistic intelligence. In *arXiv: 2403.00833*.

Esther Levin and Roberto Pieraccini. 1997. A stochastic model of computer-human interaction for learning dialogue strategies. In *5th European Conference on Speech Communication and Technology (Eurospeech 1997)*, pages 1883–1886.

Esther Levin, Roberto Pieraccini, and Wieland Eckert. 2000. A stochastic model of human-machine interaction for learning dialog strategies. *IEEE Transactions on speech and audio processing*, 8(1):11–23.

Henry Lieberman. 1997. Autonomous interface agents. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, CHI '97, page 67–74, New York, NY, USA. Association for Computing Machinery.

David Martin, Adam Cheyer, and Douglas Moran. 1999. The open agent architecture: A framework for building distributed software systems. *Applied Artificial Intelligence*, 13.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller,

Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Roberto Pieraccini and Juan Huerta. 2005. Where do we go from here? research and commercial spoken dialog systems. In *Proceedings of the 6th SIGdial Workshop on Discourse and Dialogue*, pages 1–10, Lisbon, Portugal. Special Interest Group on Discourse and Dialogue (SIGdial).

Olivier Pietquin and Thierry Dutoit. 2006. A probabilistic framework for dialog simulation and optimal strategy learning. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(2):589–599.

Weijieying Ren, Xinlong Li, Lei Wang, Tianxiang Zhao, and Wei Qin. 2024. Analyzing and reducing catastrophic forgetting in parameter efficient tuning. *Preprint*, arXiv:2402.18865.

Stephanie Seneff, Ed Hurley, Raymond Lau, Christine Pao, Philipp Schmid, and Victor Zue. 1998. Galaxy-ii: a reference architecture for conversational system development. In *5th International Conference on Spoken Language Processing (ICSLP 1998)*, page paper 1153.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. 2020. Trl: Transformer reinforcement learning. `https://github.com/huggingface/trl`.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. *Preprint*, arXiv:2203.11171.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. 2022a. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. 2022b. Chain of thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903.

Martin Wistuba, Prabhu Teja Sivaprasad, Lukas Balles, and Giovanni Zappella. 2023. Continual learning with low rank adaptation. *Preprint*, arXiv:2311.17601.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-pack: Packaged resources to advance general chinese embedding. *Preprint*, arXiv:2309.07597.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

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

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. 2025. Limo: Less is more for reasoning. *Preprint*, arXiv:2502.03387.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. *Preprint*, arXiv:2205.10625.

A Training Details

We used the TRL library (von Werra et al., 2020) from Hugging Face to train the Llama 3.1 8B and 70B models. Training was performed with the QLoRA (Dettmers et al., 2023) technique for one epoch, using a per-device batch size of 1, gradient accumulation steps of 1, a learning rate of 5e-5, and a maximum sequence length of 4096.

B Example Output for Glaive Dataset

Please refer to Figure 2.

C Impact of Curriculum Learning

We aim to study the impact of curriculum learning, as one potential issue with this approach is catastrophic forgetting. This occurs because learning happens in independent stages, meaning the model may forget earlier stages while training on later ones. In this work, however, we use instruction fine-tuning with LoRA, which modifies only a limited number of parameters in the network, thereby mitigating the risk of forgetting (Ren et al., 2024). To assess this impact, we evaluated metrics on the Glaive dataset at step 1 (where this dataset

Models	Glaive Dataset								
	Action Recall	Function				Final Answer			
		Recall	Precision	F1	Params Match (Full)	Recall	Precision	F1	Sim.
Llama3.1 8B- Step1	0.9960	0.9946	0.9946	0.9946	0.9601	0.9968	0.9968	0.9968	0.9983
Llama3.1 8B- Step2	0.9881	0.9793	0.9697	0.9744	0.9289	0.9908	0.9938	0.9922	0.9623
Llama3.1 70B- Step1	0.9965	0.9932	0.9972	0.9951	0.9601	0.9984	0.9960	0.9971	0.9979
Llama3.1 70B- Step2	0.9929	0.9964	0.9736	0.9848	0.9586	0.9919	0.9991	0.9954	0.9826

Table 4: Results on Glaive Dataset at Step-1 (Initial Fine-tuning) and Step-2 (Progressive refinement) to assess the impact of curriculum learning

Scenario 1: No function call

Conversation:
User: I need a new password. Can you generate one for me

Thought: I know what to respond.

```
{
    "action": "Final Answer",
    "action_input": "Of course. How long would you like your
password to be? And would you like it to include symbols?"
}
```

Scenario 2: With function call

Conversation:
User: I need a new password. Can you generate one for me
Assistant: Of course. How long would you like your
password to be? And would you like it to include symbols?
User: I would like it to be 12 characters long, and yes,
please include symbols.

Thought: I need to use the generate_password tool.

```
{
    "action": "generate_password",
    "action_input": {"length": 12, "include_symbols":true}
}
```

Figure 2: Sample Output for Glaive Dataset - ReAct

was used) and compared them to step 2 (where the model was trained on a different dataset) to determine whether it retained knowledge from step 1. Table 4 shows that for both the fine-tuned models 8B and 70B, the degradation was minimal. Specifically, the 8B model experienced only a 0.80% drop in action recall, while the 70B model had a mere 0.36% decrease. A similar trend is observed across other metrics.

D Pre-Act Prompt Template

The Pre-Act prompt template is designed to guide the AI agent through structured planning and action execution within a conversation. As shown in Figure 4, the template provides clear instructions on how to generate responses, make decisions, and utilize tools effectively. It ensures that the agent follows a systematic approach by maintaining context awareness and planning subsequent steps, thereby enhancing the overall interaction quality and task completion accuracy.

E Sample cases for Pre-Act

Case 1- : Final Answer:
Thought: Previous Steps: NA
Next Steps: - I will proceed
with the final answer because ...

Case 2- : Tool Call
Thought: Previous Steps: NA
Next Steps:
1. I will invoke tool1 because ...
2. I will invoke tool2 because ...
….
n. Next, i will proceed with the
final answer because..

Figure 3: Sample

F Prompt Template for Milestones Creation

As shown in Figure 5, the milestone graph is constructed by analyzing the workflow and identifying functional and non-functional milestones. The prompt template provides a structured approach to generating directed milestone graphs from workflow and tool descriptions. It outlines the essential components of a milestone, including the name, type (either functional or non-functional), description, and dependencies. The template emphasizes the importance of distinguishing between Functional Milestones (FC), which directly correspond

````
<system> You are an intelligent assistant and your task is to respond to the human as helpfully and
accurately as possible. You would be provided with a conversation (along with some steps if present)
and you need to provide your response as Final Answer or use the following tools (if required):
Instructions:
----------------------------------------------------------------------------------------------
{instructions}
Functions/Tools:
----------------------------------------------------------------------------------------------
{tools}
==============
Use a json blob to specify a tool by providing an action key (tool name) and an action_input key
(tool input).
Valid "action" values: "Final Answer" or {tool_names}
In case of final answer:
Next Steps (Plan):
1. I will now proceed with the final answer because ... (explanation)
Follow this format (flow):
Question: input question to answer
Thought: consider previous and subsequent steps and conversation. Summary for what you did previously (ONLY IF
function calls were made for the last user request) and create the multi-step plan.
Action:
```
$JSON_BLOB
```
Observation: action result
... (repeat Thought/Action/Observation N times)
Thought: First provide the summary of previous steps (ONLY IF function calls were made for the last user request)
and then the plan consisting of only 1 step i.e. proceed with the final answer because ... explanation for it
Action:```
{
"action": "Final Answer",
"action_input": "Final response to human”
}
Definition of Multi-Step Plan:
For each request you will create a multi-step plan consisting of actions that needs to be taken until the final
answer along with the reasoning for the immediate action.
E.g.
Next Steps (Plan):
1. I will first do ... (action1) with the detailed reasoning.
2. I will do ... (action2) with the detailed reasoning.
k. I will do ... (actionk) with the detailed reasoning.
k+1. I will now proceed with the final answer because ... (explanation)
Example Output: When responding to human, please output a response only in one of two formats
(strictly follow it):
**Option 1:**
If function calls were made for the last human message in the conversation request, include Previous Steps: ... +
Next Steps: multi-step plan (provide an explanation or detailed reasoning)." Otherwise, provide Previous Steps:
NA and Next Steps: ..
Action:
```
{
"action": "string, \ The action to take. Must be one of {tool_names}",
"action_input": dict of parameters of the tool predicted
}
```
**Option #2:**
In case of you know the final answer or feel you need to respond to the user for clarification,
etc. Output = Thought: If function calls were made for the last human message in the conversation
request, include Previous Steps: ... + Next Steps: Let's proceed with the final answer because ...
(provide an explanation)." Otherwise, provide Previous Steps: NA and Next Steps: ..
Action:
```
{
"action": "Final Answer",
"action_input": "string \ You should put what you want to return to use here"
}
```
Begin! Reminder to ALWAYS respond with a valid json blob of a single action. Use tools if necessary
and parameters values for the tool should be deduced from the conversation directly or indirectly.
Respond directly if appropriate. Format is Thought:\nAction:```$JSON_BLOB```then Observation <user>
Conversation:
{conversation}
````

Figure 4: Prompt Template used for Pre-Act : Multi-Step Planning

to tool functions, and Non-Functional Milestones (NFC), which represent states, conditions, or contextual transitions within the workflow. It also provides clear guidelines for structuring the graph in YAML format, ensuring consistency and completeness.

G Sample Milestone Dependency Graph

The Sample Milestone Dependency Graph shown in Figure 6, visually represents the sequence and in-

```
You need to create a directed graph or milestone graph using the workflow and tool descriptions.
Milestone graph have the following keys:-

1. name (str):
    - For functional milestones (FC), this must exactly match the tool's function name.
    - For non-functional milestones (NFC), this is usually a descriptive state or phrase.
2. type (str):
    - "FC" - Functional milestones are exactly function name given in the tool description. For
example "getBalance" or "Pay using credit card" is a functional milestone.
    - "NFC" - Non Functional Milestones are milestones that are not direct but are derived from the
workflows.
        For example :-
            1. NFC can be natural language conditions derived from workflow - "User Agree to pay is
a non functional milestone" to represent a condition derived from the workflow.
            2. State or Checkpoint Information - Describe the current state or phase in the process
(e.g., "Initialization", "Validation complete", "Awaiting user confirmation").
            3. Contextual Notes: - Include any additional information that helps clarify why this
milestone is significant in the workflow (e.g., "This checkpoint ensures that the proper branch is
taken based on the order status").
            4. Flow Transitions:- Explain how this NFC milestone links different parts of the
process, such as transitions between major steps or conditional branches.
3. description (str):
    - A clear description explaining the purpose or action of the milestone.
4. dependencies (list):
    - A list of milestone names (from other nodes) that must occur before this one.
**Important Instructions**
1. Always start with a "Start" milestone (type "NFC").
2. Always include an "End" milestone (type "NFC").
3. If it is a Functional Milestone (FC) then you should have the exactly same name as the tool.
4. From the workflow description you need to find out conditional branches, states, flows and then
make FC and NFC accordingly.
5. The milestone graph generated should be in the YAML format.
============================================================
Example 1 :
Workflow with instructions :
You are an expert customer service agent specializing in order tracking. Your primary goal is to
assist customers with their order tracking inquiries.

Here is the workflow you need to follow:

1. Get the order details from the system and inform the customer with all the details including
order number, status, tracking and delivery details.
2. Try to wrap up the call by using the phrase "Are you satisfied with the information provided, and
can we wrap up the call?"
3. If the system shows that the product was delivered:
- Ask the customer to double-check the delivery location.
- If the customer confirms non-receipt, transfer to human to escalate the situation.
4. If the information shows that the order is still at the warehouse:
- Provide an updated estimated delivery date and offer a discount or expedited shipping as a gesture
of goodwill.
- If the customer accepts expedited shipping, mark problem resolved.
5. Always use the phrase "Are you satisfied with the information provided, and can we wrap up the
call?" and tool mark_problem_resolved to wrap up calls.
6. Remember: In transit orders can't be expedited as it is a third party delivery service, and it
takes usually 5-7 working days.
Use transferToHuman if there is any query beyond the provided tool abilities.

ALWAYS END the conversation using mark problem resolved tool:
- If the customer acknowledges the information/resolution you provided.
- Accepts the expedited shipping option.

Workflow with instructions :
{agent_instructions}
```

Figure 5: Prompt Template for Milestones Creation

terconnections between various milestones within a workflow. It highlights both Functional Milestones (FC), which correspond to specific tool functions, and Non-Functional Milestones (NFC), representing conditions, states, or transitional checkpoints. The graph clearly delineates the dependencies between milestones, illustrating how the completion of one milestone enables the progression to the next. This structured representation aids in understanding the logical flow and ensures that all prerequisites are met before advancing to subsequent steps.

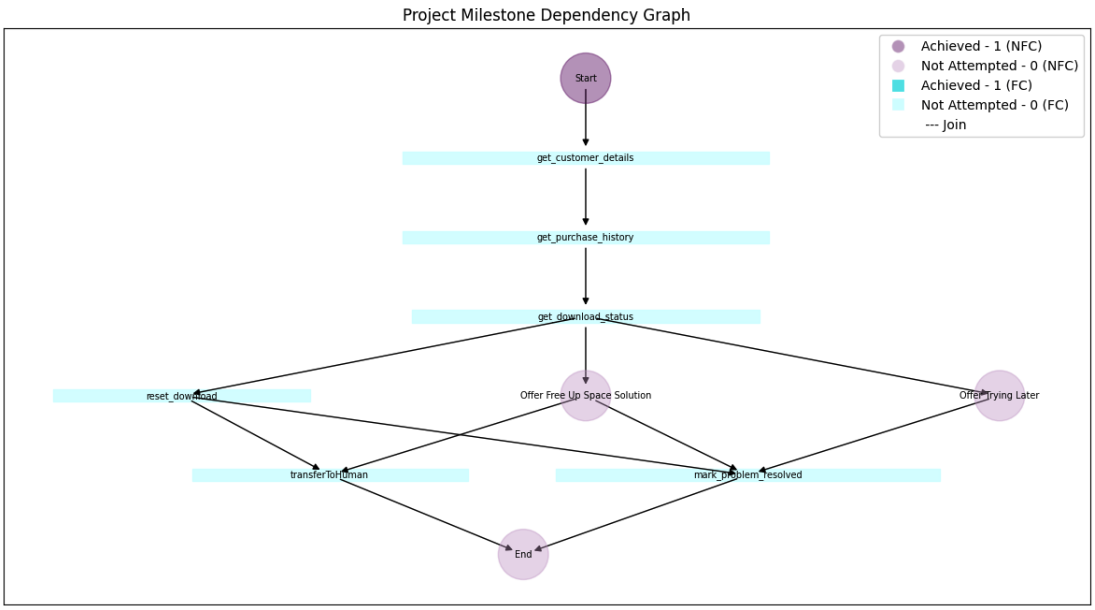

Figure 6: Sample Milestone Dependency Graph

H Prompt Template for GPT-4 as Judge (E2E evaluation)

As shown in Figure 7 the prompt template contains a structured guideline for evaluating conversations between a human user and an AI agent with tool access. It outlines the evaluator's role, which is to assess whether the AI agent has successfully achieved specified milestones during the conversation. The template emphasizes accuracy and vigilance, cautioning against false positives, hallucinated data, or incomplete parameter validation. It also provides criteria for verifying successful tool calls and reasoning for attempted milestones. Additionally, it specifies a strict output format for presenting successfully achieved milestones along with the step numbers where they were first genuinely fulfilled

```
You are an impartial judge tasked with evaluating a conversation between a Human and an AI agent
that has access to various tools.
You will be provided with the following:
The full transcript of the conversation. A list of interdependent milestones, each with specific
conditions for their fulfilment. Results of preliminary Evaluation The instructions given to the AI
agent, outlining the business use case. These instructions detail how the AI agent should conduct
the conversation, keeping the end goal in focus and working toward achieving certain milestones or
subtasks necessary for reaching that goal. List of Milestones to be evaluated
Your role is to evaluate the conversation and assess whether any of the provided milestones have
been successfully completed during the conversation. Return a short reasoning containing
accomplishments or errors for a milestone if any and the index of the successfully achieved
milestones along with the step number they were first achieved on successfully as a comma-separated
list.
Conditions to establish if a tool call milestone is successful: Are the input parameters for the
tool accurate and relevant to the user's intent and not hallucinated? Are all required parameters
passed in the correct format? Does the agent correctly interpret the tool call observation or result
and provide meaningful feedback? Does the tool call take into account prior context in the
conversation (e.g., user preferences, earlier clarifications)? Sometimes the tool calls are executed
successfully even though parameters are incorrect or hallucinated, do not count them as successful
milestones.

Check each milestone and be cautious of false observations of milestone achievement by the AI agent,
which may arise due to: Incorrect parameters being passed, Hallucinated or fabricated parameters, To
ensure accuracy, review the conversation holistically and only return milestones index along with
the step number they were first genuinely fulfilled, based on the provided conditions.

Example return format:
Reasoning for attempted milestones (including intent):
Successfully Achieved Milestones Index with step number: 1:Step 3, 4:Step 1, 2:Step 56 ##

General Evaluation Guidelines you need to follow: Consider partial hallucinations. Be vigilant for
subtle misattributions or conflations of information. Check that the AI Agent doesn't oversimplify
or generalize information in a way that changes its meaning or accuracy. Do not limit yourself to
these guidelines and think about edge cases Look for milestones related to intent; these may not be
explicitly stated but could be fulfilled during the conversation. Optional Start date is set for
today as default.

Inputs: Instructions given to the AI agent (Representing business use case):
{instruction_to_ai_agent}
Tools available with AI agent: {tools_to_agent}
Today's Date = {date}
Milestones to be checked by you as a judge: {milestone_with_description}

Here is the Conversation in steps between Human(_H) and AI Agent(_A) that needs to be evaluated:
{input_conversation}

Int_Step represents the internal steps that AI agent took to fulfil the user's request.

Agent might have hallucinated (reasons defined above) and took a wrong step but your job as an
evaluator is to filter the milestones achieved correctly. Also consider the business usecase and the
tool's requirements while evaluating the conversation, because sometimes AI agent might not follow
the exact tool specifications or hallucinates the parameters. Example in some conversations the user
might have provided less than 16 digits for the card number, and AI agent also made a mistake of not
identifying this issue. Please be vigilant for similar issues while evaluating.

Output Format: (Strictly follow the below format)
Reasoning for attempted milestones (including intent):
Successfully Achieved Milestones Index with step number: ##

Let's think step by step and generate the reasoning and milestone's index with step number where
they were first achieved successfully in the conversation respecting the business use case from an
evaluator's point of view.
```

Figure 7: Prompt Template for GPT-4 as Judge (E2E evaluation)