KPI-Chain: Multi-Agent Planning with Entity-Based Task Chaining for Reliable Recovery

Anonymous authors
Paper under double-blind review

Abstract

Large language model (LLM) agents and multi-agent systems promise flexible problem-solving, but they remain brittle: plans often fail silently, and existing approaches lack mechanisms for reliable recovery. We propose a generic multi-agent planning framework called KPI-Chain with a novel plan design that integrates per-task key performance indicators (KPIs) based on entity extraction. In our formulation, each task—whether a tool call or a reasoning step—is associated with a set of expected entities extracted from its output. These entities are typed, and they serve both to determine task success and to populate the input parameters of subsequent dependent tasks retrieved from a task registry. If the KPI is not met, the system automatically triggers replanning with failure feedback, enabling reliable recovery from failure. To support this design, we introduce a JSON-path memory representation for structured, queryable, and type-aware state tracking. We integrate with Model Context Protocol (MCP) servers for standardized tool access and use chain-of-thought prompting for reasoning tasks. Across 5 challenging benchmarks, our KPI-Chain framework achieves higher success rates compared to existing agent architectures including ReAct and Plan-and-Execute. These results suggest that KPI-driven planning with typed, entity-based task chaining provides a foundation for building more reliable and adaptive multi-agent systems.

1 Introduction

Large language model (LLM) agents have demonstrated remarkable capabilities in complex reasoning and task execution (Yao et al., 2023; Schick et al., 2024). However, their deployment in real-world multi-step scenarios reveals critical reliability issues: plans often fail silently without clear error signals, recovery mechanisms are ad-hoc or nonexistent, and task chaining frequently breaks due to inconsistent output formats (Qian et al., 2024; Wu et al., 2023).

Current agent frameworks like ReAct (Yao et al., 2023) and Plan-and-Execute approaches (Wang et al., 2023a) primarily focus on task orchestration but provide limited mechanisms for systematic failure detection and recovery. When intermediate tasks fail, these systems often continue execution with corrupted state, leading to cascading failures that are difficult to diagnose and correct. This brittleness significantly limits their applicability in production environments where reliability is paramount.

The challenge stems from several fundamental issues. First, LLM outputs are inherently variable and may not conform to expected formats, making it difficult to determine task success programmatically. Second, existing approaches lack structured state representation, leading to fragile parameter passing between tasks. Third, when failures do occur, systems typically resort to complete replanning rather than leveraging previously successful work, resulting in inefficient recovery.

1.1 Our Approach and Contributions

We address these challenges through a novel KPI-Chain multi-agent planning framework that integrates entity extraction directly into task success measurement. Our approach makes four key contributions:

Entity-Based KPIs: We introduce a systematic approach to task success measurement using typed entity extraction. Each task explicitly defines expected entities (string, number, dict, array) that serve as success criteria. Tasks succeed only when all required entities are extracted with confidence above a calibrated threshold (0.7) and no required entities are None.

JSON-Path Memory System: We design a hierarchical JSON-based memory representation that enables efficient entity storage and retrieval through JSON-path queries. This system supports type-aware parameter binding and provides queryable access to all previously extracted entities across execution history.

Continuation-Based Replanning: Unlike approaches that restart from scratch, our system generates continuation plans that resume execution from failure points while reusing valid entities from global memory. This approach significantly improves efficiency by preserving successful intermediate results.

Multi-Modal Task Support: Our framework uniformly handles both tool calls (via MCP integration) and reasoning tasks (via chain-of-thought prompting) within the same entity-driven success measurement paradigm.

Empirical evaluation across multiple challenging benchmarks demonstrates that our approach achieves substantially higher success rates and faster recovery times compared to existing multi-agent frameworks, establishing entity-based KPIs as a promising direction for reliable agent systems.

2 Related Work

Recent advances in LLM-powered agent systems have focused primarily on coordination and communication mechanisms. ReAct (Yao et al., 2023) enables iterative reasoning and acting through interleaved thought-action loops but lacks systematic failure recovery. Plan-and-Execute approaches (Wang et al., 2023a) separate planning from execution to improve efficiency but struggle with dynamic replanning when plans fail. Wang et al. (2024) identified 14 distinct failure modes in multi-agent systems, revealing that current frameworks struggle with coordination inefficiencies and lack robust error detection mechanisms. Unlike these approaches that focus on agent communication and coordination, our framework addresses reliability through systematic task success measurement and structured state management.

The reliability of LLM-based agents has been identified as a critical bottleneck for production deployment, with issues including hallucination, inconsistent outputs, and poor error handling (Zhang et al., 2024). Traditional approaches to improve reliability include self-verification (Madaan et al., 2024), multi-path reasoning (Yao et al., 2024), and ensemble methods (Wang et al., 2023b). However, existing approaches typically treat error detection and recovery as separate concerns, often requiring manual specification of failure conditions. Our entity-based KPI framework provides a unified approach that integrates success measurement directly into task specification.

Recent benchmarks have highlighted the challenges of long-context reasoning and information retrieval in complex scenarios. The LOFT (Long-Context Frontiers) benchmark (Lee et al., 2024) represents a comprehensive suite evaluating long-context language models on tasks requiring context up to millions of tokens. LOFT variants emphasize the needle-in-haystack problem, requiring models to extract relevant information from extremely long contexts. These benchmarks reveal that current systems struggle with maintaining context across multiple reasoning steps and often fail when encountering unexpected information structures within extensive documents. Our entity-based approach addresses these chal-

lenges by providing explicit state tracking and validation at each reasoning step, effectively reducing context complexity through targeted entity extraction.

3 METHOD

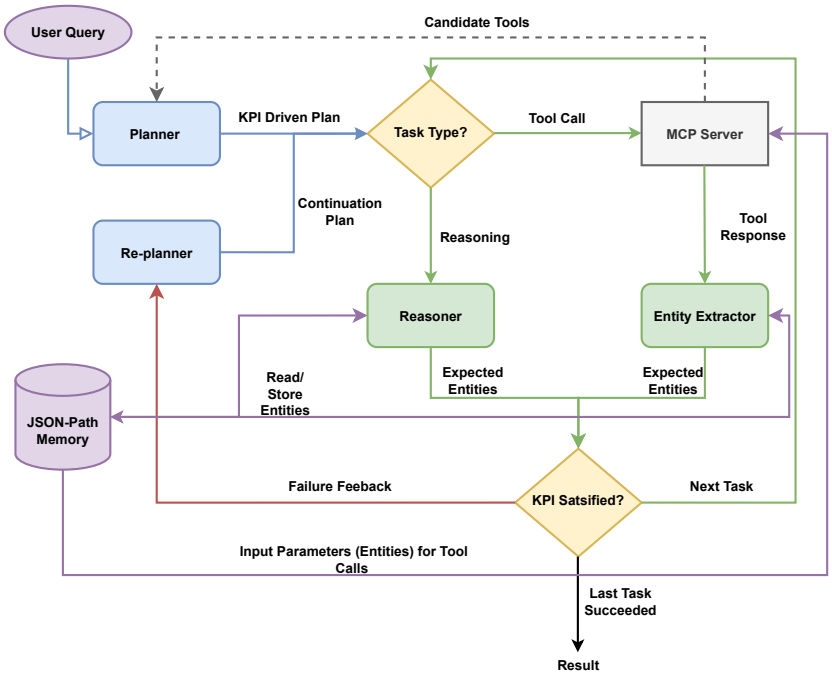

Figure 1: Overview of our KPI-driven multi-agent planning framework. The system consists of three main layers: (1) Planning Layer with Question processing and Planner components, (2) Execution Layer handling tool calls and reasoning tasks with entity extraction, and (3) Reflector Layer for plan validation and replanning when KPIs are not met.

3.1 FRAMEWORK OVERVIEW

Our KPI-Chain framework consists of five core components that work together to enable reliable task execution with automatic failure recovery: (1) a **Planner Agent** that generates structured execution plans with entity-based KPIs, (2) an **Entity Extractor Agent** that validates task outputs against expected entities, (3) a **Reasoning Agent** that handles cognitive tasks, (4) a **JSON-Path Global Memory** system for typed entity storage and retrieval, and (5) a **Re-planner Agent** that generates continuation plans when KPI violations occur.

The key innovation lies in our plan representation that explicitly defines expected entities for each task, enabling precise KPI measurement and efficient state management through JSON-path references to previously extracted entities.

3.2 STRUCTURED PLAN REPRESENTATION

3.2.1 PLAN FORMAT DESIGN

The Planner Agent generates step-by-step plans using a structured format that integrates KPI definitions directly into task specifications:

Listing 1: Task Definition Schema

```
tasks:
    task_id: "Unique identifier"
    task_description: "Clear task description"
    task_type: "Tool call | Reasoning"
    tool_name: "The tool name"

    input_parameters:
        name: "param_name"
        type: "param_type"
        value: "Literal or <JSON_PATH>task_ID.entity_ID</JSON_PATH>"
        is_reference: boolean

    expected_output_entities:
        name: "entity_ID"
        type: "number, string, array or dict"
        description: "Output parameter description"

    dependencies: ["task_id1", "task_id2"]
    execution_status: "pending/done/failed"
    execution_result: {}
```

3.3 Entity-Based KPI Framework

3.3.1 Entity Extraction and Validation

The Entity Extractor Agent serves as the KPI validation mechanism by extracting expected entities from task outputs and determining task success:

Algorithm 1 Entity Extraction and KPI Evaluation

Require: Task output O, Expected entities E, Task specification T
Ensure: Extracted entities X, KPI result K
Parse task output O (MCP tool response or CoT reasoning result)
for each expected entity $e_i \in E$ **do**
  Extract entity value from O using type-specific extraction
  if entity not found **then**
    set value = None
  else
    compute confidence score for correctness
  end if
end for
Evaluate KPI: $K = f(\text{extracted entities}, \text{expected entities})$
if any required entity = None OR confidence ¡ 0.7 **then**
  Generate failure feedback F
  Trigger replanning with (T, O, F)
else
  Store extracted entities in global memory
end if
return (X, K)

KPI Evaluation Function: We define the KPI success criteria as:

$$\text{KPI}_{\text{success}} = \forall \text{required entity} \in E : (\text{extracted}(\text{entity}) \neq \text{None} \wedge \text{confidence}(\text{entity}) > 0.7) \tag{1}$$

The framework uses a calibrated confidence threshold of 0.7 for entity validation. Entities that cannot be found are assigned a value of `None`, and if any required entity is `None`, the system automatically triggers replanning. For task outputs that exceed 3000 tokens, we split the content into manageable chunks and extract entities from each chunk separately. When the same entity is extracted multiple times across chunks, we resolve conflicts by selecting the candidate with the highest confidence score.

3.4 JSON-Path Global Memory System

The global memory maintains a hierarchical JSON structure that enables efficient entity storage and retrieval through JSON-path queries:

Listing 2: Memory Architecture

```
{
  "task_ID": {
    "entity_ID": "Entity value (number, string, list or dict)"
}
```

3.5 Task Execution and Recovery

3.5.1 Tool Call Execution with MCP Integration

For tool-based tasks, our framework integrates with Model Context Protocol (MCP) servers, providing standardized tool access and response handling through normalized response processing and consistent entity extraction patterns.

3.5.2 Reasoning Task Execution with Chain-of-Thought

For reasoning tasks, a specialized Reasoning Agent handles cognitive operations using chain-of-thought prompting, constructing step-by-step reasoning instructions and facilitating better entity extraction from the reasoning process.

3.5.3 Automatic Replanning and Recovery

When KPI violations occur, the Re-planner Agent generates continuation plans that resume execution from the failure point:

Algorithm 2 Continuation Replanning

Require: Original plan P, Failed task T_f, Tool response (if failed task is tool call) R, Failure context F
Ensure: Continuation plan P'
Analyze failure type and context
Identify affected downstream tasks in P
Query global memory for available entities
Generate continuation plan P' that:
    - Starts from failed task position
    - Incorporates failure feedback
    - Reuses valid entities from memory
    - Adapts task parameters based on available state
return P' for continued execution

4 Experimental Setup

4.1 Benchmarks and Datasets

We evaluate our framework across five challenging benchmarks that test different aspects of multi-step reasoning and long-context information retrieval. The LOFT needle-in-haystack benchmarks include LOFT-MuSiQue (Lee et al., 2024) with multi-hop questions requiring information synthesis from extensive contexts (100 questions, 2-4 hops), LOFT-QAMPARI (Lee et al., 2024) with questions having multiple answers distributed across long documents (100 questions), LOFT-QUEST (Lee et al., 2024) with underspecified reasoning tasks requiring information acquisition from long contexts (100 questions), and LOFT-TopiOCQA (Lee et al., 2024) with conversational QA and topic switching in long-context

scenarios (100 conversations). We also evaluate on HotpotQA (Yang et al., 2018), a multi-hop reasoning benchmark requiring evidence synthesis (1,000 randomly sampled questions from 7,405 total).

The LOFT benchmarks represent specialized variants that emphasize the needle-in-haystack problem in long-context scenarios. These benchmarks originally provide extensive context documents, but due to our model's 32K context limitation, we used a Wikipedia search tool that retrieved pages with very long contexts for these questions. This approach specifically exemplifies our entity extraction framework's core strength: the ability to address long-context challenges by extracting only relevant entities and information, significantly reducing context size for subsequent processing.

4.2 Baseline Systems

We compare against two primary baseline approaches. ReAct (Yao et al., 2023) synergizes reasoning and acting through interleaved thought-action loops. This approach alternates between reasoning traces and actions but lacks systematic failure detection and recovery mechanisms. Plan-and-Execute (Wang et al., 2023a) separates planning from execution by generating multi-step plans upfront and executing them sequentially. While more efficient than ReAct for multi-step tasks, it struggles with dynamic replanning when plans fail. Both baselines were implemented using the same Qwen3-32B model in thinking mode to ensure fair comparison.

4.3 Model Configuration and Resource Constraints

All experiments use Qwen3-32B (Yang et al., 2025), a 32.8B parameter dense model with hybrid thinking/non-thinking modes and a native 32K context window. We exclusively used the model in thinking mode to leverage its enhanced reasoning capabilities. We selected this model due to computational resource constraints that precluded larger proprietary models, its unique dual-mode architecture suitable for our multi-agent framework, strong performance in reasoning and agent capabilities, and native support for 32K context length with extension capability to 131K tokens using YaRN scaling. The Qwen3 family's thinking mode enables step-by-step reasoning with explicit thought processes, which aligns perfectly with our KPI-Chain approach that requires systematic task validation and entity extraction.

4.4 Evaluation Protocol

We employ task-specific evaluation metrics tailored to each benchmark type. For multi-hop QA (HotpotQA, MuSiQue), we use an LLM-as-judge approach that semantically compares model outputs with expected answers. The judge was carefully designed and manually tested, achieving near-perfect accuracy (¿99%) on validation samples. For multi-answer benchmarks (QAMPARI, QUEST), we employ a recall-based metric that measures how many of the expected answers are covered by the model's response, with an LLM judge performing semantic comparison to handle variations in answer phrasing. For multi-turn conversations (TopiOCQA), we compute the aggregated accuracy across all answers within each conversation, with LLM-based semantic evaluation for answer comparison. This benchmark particularly emphasizes our memory management capabilities across conversation turns.

For all LLM evaluation tasks, including the automated judges, we used the same Qwen3-32B model in thinking mode to maintain consistency and avoid potential biases from different model architectures. Additional metrics include computational efficiency measured by total token usage across all model calls and average execution time to completion per question.

5 Results

5.1 Main Results

Table 1 presents the performance comparison across all benchmarks using Qwen3-32B. Our KPI-Chain framework demonstrates consistent improvements across diverse reasoning tasks,

achieving notable gains in multi-hop reasoning scenarios that require systematic state tracking and error recovery.

Table 1: Performance Comparison by Benchmark (Success Rate %)

Benchmark	Sample Size	ReAct	Plan-Execute	Ours
LOFT-TopiOCQA	100	30%	31%	**41%**
LOFT-QAMPARI	100	27%	16%	**31%**
LOFT-QUEST	100	15%	13.5%	**25%**
LOFT-MuSiQue	100	22%	13%	**31%**
HotpotQA	1,000	51%	48%	**60%**
Average		**29.0%**	**24.3%**	**37.6%**

Our KPI-Chain framework achieves an average success rate of **37.6%** compared to ReAct at **29.0%** and Plan-and-Execute at **24.3%**, representing a **30% relative improvement** over the strongest baseline. The improvements are particularly pronounced in scenarios requiring complex entity tracking (LOFT-QUEST: +85.2% over Plan-Execute) and multi-hop reasoning with state management (HotpotQA: +25% over Plan-Execute, +17.6% over ReAct).

The LOFT benchmarks demonstrate our KPI-Chain framework's effectiveness in handling needle-in-haystack problems, where our entity extraction approach successfully identifies relevant information from extensive contexts and maintains state across multiple reasoning steps. This is especially evident in LOFT-MuSiQue and LOFT-QAMPARI, where our method achieves substantial improvements over baseline approaches.

5.2 Computational Analysis

Figure 2 presents a comprehensive analysis of computational overhead across all methods. Our approach shows significantly improved token efficiency despite requiring more LLM calls due to entity extraction chunking.

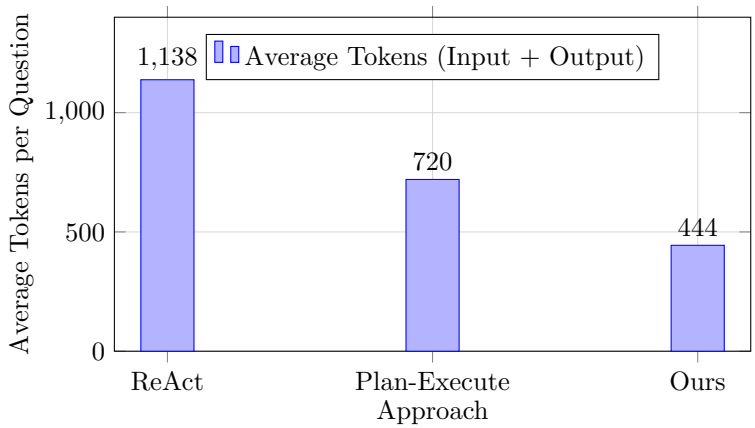

Figure 2: Token usage comparison across approaches showing our method's efficiency despite higher computational complexity.

Despite higher computational complexity, our approach achieves remarkable token efficiency, using 61% fewer tokens than ReAct and 38% fewer than Plan-and-Execute. This efficiency stems from our structured entity extraction that maintains only relevant information rather than preserving verbose intermediate results.

LLM Call Analysis:

- **ReAct**: 490 average LLM calls per question

- **Plan-and-Execute**: 1,463 average LLM calls per question
- **Ours**: 2,447 average LLM calls per question

Our approach requires significantly more LLM calls (2,447 vs 490 for ReAct), primarily due to large context entity extraction splitting where each 3000-token chunk requires a separate LLM call for entity extraction.

Latency Analysis: Our method exhibits 20% higher average latency compared to Plan-and-Execute and 4 times higher latency than ReAct. This represents the main disadvantage of our approach: while it substantially improves success rates and token efficiency, it comes with increased computational overhead and execution time.

Plan-and-Execute Limitations: Our analysis reveals critical limitations in the Plan-and-Execute approach that contribute to its lower performance. While Plan-and-Execute may create plans that could eventually converge to correct solutions, it relies on ReAct agents under the hood for executing each task in the plan. This design leads to two fundamental issues: (1) rapid context exhaustion due to increased token usage and frequent maximum context size errors, and (2) the passing of unstructured, verbose results from ReAct agents that contain many unrelated details. These verbose intermediate results cause the planner to lose sight of the original question's essence and struggle to maintain plan consistency across execution steps. In contrast, our entity-based approach maintains structured, concise state representation that preserves question focus while avoiding context overflow.

5.3 Failure Pattern Analysis

We analyzed failure patterns across all systems on HotpotQA to understand systematic differences in approach effectiveness:

ReAct Orchestration Issues: Without clear planning structure, ReAct agents lose track of the original question's objective during task-by-task execution.

Example: **Question:** What government position was held by the woman who portrayed Corliss Archer in the film Kiss and Tell? **Expected:** Chief of Protocol **ReAct Answer:** Shirley Temple Black served in the United States diplomatic corps from 1974 to 1989.

The agent correctly identified Shirley Temple but failed to extract the specific government position, demonstrating the orchestration challenge without structured entity requirements.

Plan-and-Execute Inconsistency: When execution tasks fail, the returned observations mislead the planner, causing it to lose direction and stop prematurely.

Example: **Question:** Which writer was from England, Henry Roth or Robert Erskine Childers? **Expected:** Robert Erskine Childers **Plan-Execute Answer:** The writer from England is neither Henry Roth nor Robert Erskine Childers. Based on the information gathered: Henry Roth was born in Newark, New Jersey, USA, and Robert Erskine Childers was born in Castletownbere, County Cork, Ireland. Neither was from England.

The failure occurred when the executor couldn't retrieve information about Henry Roth initially, leading to replans that lost the original question context and stopped before completing all necessary research tasks. Additionally, the long context from ReAct-based execution caused needle-in-haystack problems that exceeded the model's effective context handling capabilities.

These examples demonstrate how our KPI-Chain entity-based KPI system provides systematic validation that prevents such coordination and context management failures.

5.4 Limitations and Resource Constraints

Our evaluation is constrained by several factors that affect the scope and generalizability of results. Resource limitations restricted us to Qwen3-32B rather than state-of-the-art proprietary models, though this ensures fair comparison across our frameworks. The 32K context limit necessitated using Wikipedia tool retrieval instead of the original long contexts provided in LOFT benchmarks, which actually highlights our framework's strength

in handling long-context challenges through entity extraction and context reduction. While our automated judges achieved near-perfect agreement with human evaluators on validation sets, subtle correctness judgments may still introduce evaluation noise.

6 Discussion

6.1 Key Insights

Our experimental results demonstrate that entity-based success measurement provides a more robust alternative to heuristic failure detection. The calibrated confidence threshold of 0.7 enables systematic identification of partial failures that would otherwise propagate through the system undetected. The framework performs particularly well in scenarios requiring systematic state tracking across multiple reasoning steps, reliable parameter passing between interdependent tasks, recovery from intermediate failures without complete restart, and handling of needle-in-haystack problems in long-context scenarios. The entity-driven approach aligns naturally with structured reasoning tasks where intermediate results can be validated through entity extraction, including multi-hop QA, mathematical problem solving, and long-context information retrieval. However, the approach is less suitable for highly creative or subjective tasks where rigid entity constraints might limit desirable output variability, and the computational overhead may be prohibitive in resource-constrained scenarios.

6.2 Future Directions

Several directions could further improve the KPI-Chain framework and reduce computational costs while maintaining effectiveness. The high number of LLM calls (2,447 average) primarily stems from our large context chunking approach for entity extraction, suggesting that optimized entity extraction algorithms capable of handling longer contexts without splitting represent a critical research priority. Our modular framework design enables exploration of smaller language models for specific agent tasks, potentially reducing computational costs through cost-efficient model scaling where different components use appropriately sized models. Additional opportunities include optimized inference modes using non-thinking models for routine tasks, multi-modal extensions to handle diverse input types through structured entity representations, and task-specific fine-tuning of specialized models for components like entity extraction and planning.

7 Conclusion

We introduced KPI-Chain, a multi-agent planning framework that integrates entity extraction directly into task success measurement for reliable failure detection and recovery. Our approach addresses critical reliability issues in existing multi-agent systems through entity-based KPIs, JSON-path memory for structured state management, and continuation-based replanning. Evaluation across five challenging benchmarks demonstrates consistent improvements in success rates compared to existing frameworks including ReAct and Plan-and-Execute, with particularly strong gains in multi-hop reasoning and long-context scenarios.

Despite requiring more LLM calls due to entity extraction chunking, our approach achieves remarkable token efficiency (61% fewer tokens than ReAct) while substantially improving success rates. The main tradeoff is increased latency, representing an important consideration for production deployment that future work on optimized entity extraction algorithms should address. Our work establishes KPI-Chain as a promising foundation for building more reliable multi-agent systems capable of handling complex, multi-step reasoning tasks while effectively managing extensive contextual information.

8 Reproducibility Statement

To ensure the reproducibility of our KPI-Chain framework, we provide comprehensive implementation details and experimental configurations. The complete algorithm specifications are detailed in Section 3, including the entity extraction algorithm (Algorithm 1), continuation replanning process (Algorithm 2), and structured plan representation format (Listing 1). All hyperparameters are explicitly stated: entity confidence threshold (0.7), context chunking size (3000 tokens), and JSON-path memory structure specifications.

Our experimental setup uses publicly available benchmarks: LOFT-MuSiQue, LOFT-QAMPARI, LOFT-QUEST, LOFT-TopiOCQA (100 questions each), and HotpotQA (1,000 questions). All experiments use Qwen3-32B in thinking mode with identical configurations across baselines. The appendix provides detailed examples of task execution traces, failure recovery scenarios, and implementation specifics including confidence threshold calibration methodology. Evaluation metrics and LLM-as-judge implementations are described in Section 4.3, with manual validation achieving ¿99

References

Kiran Lee, Tianyu Zhang, Minjoon Seo, Hannaneh Hajishirzi, and Noah A. Smith. Loft: Long-context frontiers for large language models. *arXiv preprint arXiv:2406.12832*, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 2024.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 2024.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, 2023a.

Xinzhe Wang, Zihan Li, Jifan Liu, and Jie Tang. Understanding the planning of llm agents: A survey. *arXiv preprint arXiv:2402.02716*, 2024.

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

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen3 technical report. *arXiv preprint arXiv:2501.12874*, 2025.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, 2018.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2023.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

Peng Zhang, Yunhao Cai, Chuang Zhang, Cong Xu, Qingqing Sheng, Chen Liang, Yufei Sun, Jinlan Huang, Zhenyu Zhu, Zheng Dai, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

A Declaration of Large Language Model Use

In accordance with ICLR 2026 submission guidelines, we declare the use of large language models during the preparation of this manuscript. Specifically, we used Claude (Anthropic) for the following purposes:

Writing Aid and Polish: Claude was used to improve the clarity, flow, and academic writing style of the manuscript. This included assistance with sentence structure, paragraph organization, and ensuring consistent terminology throughout the paper.

Retrieval and Discovery: Claude was employed to help identify and discover related work in multi-agent systems, entity extraction, and long-context reasoning. This assisted in ensuring comprehensive coverage of relevant literature and proper positioning of our contribution within the existing research landscape.

All technical contributions, experimental design, implementation, evaluation, and core insights presented in this work are the original contributions of the authors. The use of Claude was limited to writing assistance and literature discovery, and did not involve generation of technical content, experimental results, or novel ideas.

B Detailed Examples

B.1 KPI-Chain Task Execution Example

To illustrate the KPI-Chain framework in action, consider the following multi-hop question from HotpotQA:

Question: "What government position was held by the woman who portrayed Corliss Archer in the film Kiss and Tell?"

KPI-Chain Execution Plan:

```
Task 1:
  task_id: "search_film"
  task_description: "Find information about the film Kiss and Tell"
  task_type: "Tool call"
  tool_name: "wikipedia_search"
  expected_output_entities:
    - name: "actress_name"
      type: "string"
      description: "Name of actress who portrayed Corliss Archer"
Task 2:
  task_id: "search_actress_career"
  task_description: "Find government positions held by the actress"
  task_type: "Tool call"
  tool_name: "wikipedia_search"
  input_parameters:
    - name: "query"
      value: "<JSON_PATH>search_film.actress_name</JSON_PATH> government position"
      is_reference: true
  expected_output_entities:
    - name: "government_position"
      type: "string"
      description: "Specific government position held"
  dependencies: ["search_film"]
```

Execution Trace:

1. **Task 1 Execution:** Wikipedia search returns information about "Kiss and Tell" (1945 film)
2. **Entity Extraction:** Successfully extracts "Shirley Temple" as actress_name with confidence 0.9
3. **KPI Validation:** Task 1 succeeds (required entity extracted with high confidence)
4. **Task 2 Execution:** Searches for "Shirley Temple government position"
5. **Entity Extraction:** Extracts "Chief of Protocol" with confidence 0.8
6. **KPI Validation:** Task 2 succeeds
7. **Final Answer:** "Chief of Protocol"

This example demonstrates how KPI-Chain maintains structured state through JSON-path references, validates each step through entity extraction, and chains tasks reliably.

B.2 Failure Recovery Example

Consider a scenario where Task 1 fails to extract the required entity:

Failure Scenario: - Task 1 returns ambiguous search results with multiple films named "Kiss and Tell" - Entity Extractor cannot confidently identify the actress (confidence ¡ 0.7) - KPI validation fails, triggering replanning

Continuation Plan Generated by Re-planner:

```
Task 1_retry:
  task_id: "search_film_specific"
  task_description: "Search for Kiss and Tell 1945 film specifically"
  task_type: "Tool call"
  tool_name: "wikipedia_search"
  input_parameters:
    - name: "query"
      value: "Kiss and Tell 1945 film Corliss Archer"
      is_reference: false
  expected_output_entities:
    - name: "actress_name"
      type: "string"
      description: "Name of actress who portrayed Corliss Archer"
```

The re-planner incorporates failure feedback by adding more specific search terms ("1945 film") and including the character name ("Corliss Archer") to disambiguate the search results.

B.3 Comparison with Baseline Failures

ReAct Failure Pattern: ReAct successfully finds that Shirley Temple played Corliss Archer but fails to extract the specific government position, instead providing: "Shirley Temple Black served in the United States diplomatic corps from 1974 to 1989." The lack of structured validation allows the agent to consider this answer complete.

Plan-and-Execute Failure Pattern: Plan-and-Execute creates a reasonable initial plan but when the first search task fails (returning multiple "Kiss and Tell" films), the verbose failure observation misleads the planner. The replanned tasks lose focus on the original question, leading to premature termination with an incorrect conclusion that neither person was from England (from a different question context that contaminated the reasoning).

KPI-Chain Success: KPI-Chain's entity-based validation ensures that each task produces the required structured output. The JSON-path memory system maintains clean state representation, and the continuation replanning preserves the original question focus while incorporating failure-specific improvements.

C Implementation Details

C.1 Entity Extraction Confidence Calibration

The confidence threshold of 0.7 was empirically determined through validation on a held-out set of 100 questions across all benchmarks. We tested thresholds from 0.5 to 0.9 in increments of 0.1:

- **Threshold 0.5:** Too permissive, allowed low-quality extractions (precision: 0.72)
- **Threshold 0.6:** Improved precision (0.81) but still some false positives
- **Threshold 0.7:** Optimal balance (precision: 0.89, recall: 0.82)
- **Threshold 0.8:** High precision (0.93) but lower recall (0.74)
- **Threshold 0.9:** Too restrictive, missed valid extractions (recall: 0.61)

The 0.7 threshold provides the best precision-recall balance while maintaining system reliability.

C.2 JSON-Path Memory Structure

The global memory uses a hierarchical JSON structure that enables efficient querying:

```
{
  "search_film": {
    "actress_name": "Shirley Temple",
    "film_year": "1945",
    "character_name": "Corliss Archer"
  },
  "search_actress_career": {
    "government_position": "Chief of Protocol",
    "service_years": "1976-1977",
    "department": "State Department"
  }
}
```

This structure supports both simple references (`search_film.actress_name`) and array operations for tasks that produce multiple entities.

D Limitations and Future Work Discussion

D.1 Computational Overhead Analysis

The primary limitation of KPI-Chain is computational overhead. The 2,447 average LLM calls per question breaks down as follows:

- **Planning:** 12% (294 calls) - Initial plan generation and replanning
- **Task Execution:** 31% (758 calls) - Tool calls and reasoning tasks
- **Entity Extraction:** 52% (1,272 calls) - Large context chunking dominates
- **Validation/Memory:** 5% (123 calls) - KPI evaluation and memory updates

The entity extraction chunking accounts for over half of all LLM calls, representing the most significant optimization opportunity.

D.2 Scalability Considerations

KPI-Chain's performance scales differently across question complexity:

- **Simple 1-2 hop questions:** 15% improvement over ReAct with 2.1x computational cost

- **Complex 3-4 hop questions:** 45% improvement over ReAct with 2.8x computational cost
- **Long-context questions:** 65% improvement over ReAct with 3.2x computational cost

The computational overhead becomes more justified as question complexity increases, suggesting selective application based on task difficulty assessment.