

000 KPI-CHAIN: MULTI-AGENT PLANNING WITH 001 ENTITY-BASED TASK CHAINING FOR RELIABLE RE- 002 COVERY 003

006 **Anonymous authors**

007 Paper under double-blind review

011 ABSTRACT

013 Planning-based LLM agent frameworks promise flexible problem-solving
 014 through structured task decomposition, but they remain brittle: plans of-
 015 ten fail silently, and existing approaches lack mechanisms for reliable re-
 016 covery. We propose KPI-Chain, a multi-agent planning framework with
 017 a novel plan structure that embeds per-task key performance indicators
 018 (KPIs) based on typed entity extraction. This plan design—our core con-
 019 tribution—fundamentally improves agent reliability by making task speci-
 020 fications more precise and explicit upfront. In our formulation, each task
 021 explicitly defines expected entities (string, number, array, dict) to be ex-
 022 tracted from its output. This structure drives multiple benefits throughout
 023 the system: it forces clearer, more specific task definitions during planning;
 024 it focuses extraction on only the relevant key information from tool re-
 025 sponds rather than verbose outputs; it enables reasoning tasks to produce
 026 structured, targeted results; it supports efficient and precise memory man-
 027 agement through typed entity storage; and critically, it makes failure root
 028 causes immediately identifiable when expected entities cannot be extracted.
 029 When KPIs are not met, the system automatically triggers continuation-
 030 based replanning with explicit failure feedback. To operationalize this plan
 031 structure, we introduce complementary components: an entity extractor
 032 for validating KPIs, a JSON-path memory system for typed entity storage
 033 and retrieval, MCP integration for standardized tool access, and chain-of-
 034 thought prompting for reasoning tasks. Across 5 challenging benchmarks,
 035 our KPI-Chain framework achieves higher success rates compared to ex-
 036 isting agent architectures including ReAct and Plan-and-Execute. These
 037 results demonstrate that embedding entity-based KPIs directly into plan
 038 structure provides a foundation for building more reliable and adaptive
 039 LLM agent systems.

040 1 INTRODUCTION

041 Planning-based LLM agent frameworks have demonstrated remarkable capabilities in com-
 042 plex reasoning and task execution through structured task decomposition (Yao et al., 2023;
 043 Schick et al., 2024). However, their deployment in real-world multi-step scenarios reveals
 044 critical reliability issues: plans often fail silently without clear error signals, recovery mech-
 045 anisms are ad-hoc or nonexistent, and task chaining frequently breaks due to inconsistent
 046 output formats (Qian et al., 2024; Wu et al., 2023). These failures stem not merely from
 047 execution problems, but fundamentally from how plans themselves are structured and spec-
 048 ified.

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

054 The challenge stems from several fundamental issues with existing plan representations.
 055 First, tasks are typically defined with natural language descriptions that lack explicit suc-
 056 cess criteria, making it difficult to determine programmatically whether a task has truly
 057 succeeded. Second, without structured specifications of what each task should produce,
 058 systems cannot validate outputs or manage state systematically, leading to fragile par-
 059 ameter passing between tasks. Third, when failures do occur, the absence of explicit failure
 060 indicators forces systems to resort to complete replanning rather than targeted recovery,
 061 resulting in inefficient use of previously successful work.

062 1.1 OUR APPROACH AND CONTRIBUTIONS

063 We address these challenges by introducing a novel plan structure that embeds per-task key
 064 performance indicators (KPIs) based on typed entity extraction directly into task specifi-
 065 cations. Our key insight is that explicitly defining expected entities (string, number, dict,
 066 array) for each task makes specifications more precise upfront and enables systematic vali-
 067 dation throughout execution. A task succeeds only when all required entities are successfully
 068 extracted.

069 This plan structure drives multiple benefits: **(1) Precise task specifications** by forcing
 070 explicit output entity definitions upfront, **(2) Focused information extraction** by con-
 071 straining extraction to relevant key information from tool responses, **(3) Structured rea-
 072 soning outputs** enabling clean parameter passing, **(4) Efficient memory management**
 073 through typed entities with JSON-path references, and **(5) Immediate failure diagnosis**
 074 providing explicit, actionable feedback about what failed and why.

075 To operationalize this plan structure, we introduce complementary components: an Entity
 076 Extractor Agent that validates outputs against expected entities, a JSON-Path Global
 077 Memory System for typed entity storage and retrieval, a Re-planner Agent that generates
 078 continuation plans using explicit failure feedback, Model Context Protocol (MCP) integra-
 079 tion for standardized tool access, and a Reasoning Agent using chain-of-thought prompting
 080 to produce structured entity outputs.

081 Empirical evaluation across multiple challenging benchmarks demonstrates substantially
 082 higher success rates compared to existing frameworks, establishing entity-based KPIs em-
 083 bedded in plan structure as a promising direction for reliable agent systems.

084 2 RELATED WORK

085 Recent advances in LLM agent frameworks have focused on coordination mechanisms, action
 086 representations, and recovery strategies, but have overlooked how plan structure itself drives
 087 reliability improvements.

088 **Plan Representation and Task Orchestration.** ReAct (Yao et al., 2023) enables it-
 089 erative reasoning and acting through interleaved thought-action loops but lacks structured
 090 task specifications with explicit success criteria. Plan-and-Execute approaches (Wang et al.,
 091 2023a) separate planning from execution but rely on natural language descriptions without
 092 formal output specifications, leading to fragile parameter passing and ambiguous success
 093 determination. Wang et al. (2024b) identified 14 failure modes in multi-agent systems, re-
 094 vealing that frameworks struggle with coordination precisely because they lack structured
 095 mechanisms for validating intermediate outputs. Unlike these approaches that treat task
 096 specification as an afterthought, our framework makes plan structure the central mechanism
 097 for reliability by embedding typed entity-based KPIs directly into task definitions.

098 **Action Representation and Execution.** CodeAct (Wang et al., 2024a) proposes ex-
 099 ecutable Python code as a unified action space, demonstrating improved flexibility and
 100 composition. While CodeAct addresses how actions are represented and executed, it does
 101 not tackle what each action should produce or how to systematically validate success. Our
 102 entity-based KPI structure is complementary: regardless of whether actions execute as code,
 103 JSON, or tool calls, our plan structure explicitly defines expected typed entities for success
 104 determination.

108 **Validation and Recovery.** Reliability issues including hallucination, inconsistent outputs,
 109 and poor error handling remain critical bottlenecks (Zhang et al., 2024). Traditional
 110 approaches include self-verification (Madaan et al., 2024), multi-path reasoning (Yao et al.,
 111 2024), and ensemble methods (Wang et al., 2023b). SagaLLM (Chang & Geng, 2025) intro-
 112 duces transactional guarantees with compensation mechanisms and independent validation,
 113 but operates on existing task specifications without changing how tasks define success cri-
 114 teria. Our approach is complementary: embedding entity-based KPIs into plan structure
 115 provides explicit, actionable failure signals that could enhance transaction-based systems
 116 by making validation targets concrete and machine-verifiable from the outset. Unlike ap-
 117 proaches that treat error detection and recovery as separate concerns, our entity-based KPIs
 118 integrate success measurement directly into task specification.

119 **Long-Context Reasoning.** The LOFT benchmark (Lee et al., 2024) evaluates models on
 120 needle-in-haystack problems requiring extraction from extremely long contexts, revealing
 121 that systems struggle to maintain context across reasoning steps. Our entity-based plan
 122 structure directly addresses this: by explicitly specifying which entities to extract at each
 123 step, our approach filters verbose responses to only relevant typed information, reducing
 124 context overhead while maintaining precise state tracking through JSON-path memory.
 125 This transforms long-context problems into structured state management, enabling efficient
 126 handling without overwhelming context windows.

127 Unlike existing frameworks focusing on coordination, action representations, or recovery
 128 mechanisms, our work establishes that embedding entity-based KPIs into plan structure
 129 provides a foundational mechanism for reliability: precise task specifications emerge nat-
 130 urally, validation becomes systematic rather than heuristic, and failure recovery leverages
 131 explicit knowledge of what succeeded and failed.

133 3 METHOD

135 Our KPI-Chain framework centers on a novel plan structure that embeds per-task entity-
 136 based KPIs directly into task specifications. This structure fundamentally changes how
 137 planning systems define, execute, and validate tasks. We describe the overall framework ar-
 138 chitecture (Section 3.1), then detail the plan structure design (Section 3.2), followed by how
 139 entity-based KPIs enable systematic validation (Section 3.3), structured state management
 140 through memory (Section 3.4), and reliable recovery through continuation-based replanning
 141 (Section 3.5).

143 3.1 FRAMEWORK OVERVIEW

145 Figure 1 illustrates the complete KPI-Chain architecture, which consists of three main
 146 layers designed to support and leverage the entity-based plan structure: Planning Layer:
 147 The Planner Agent generates structured execution plans where each task explicitly de-
 148 fines expected output entities with their types (string, number, array, dict). This upfront
 149 specification forces precise task definitions and enables systematic validation throughout
 150 execution. Execution Layer: Tasks are executed according to their type—tool calls invoke
 151 MCP servers while reasoning tasks use chain-of-thought prompting. Critically, the Entity
 152 Extractor Agent validates all tool call outputs against the expected entities defined in the
 153 plan structure, determining KPI success based on a calibrated confidence threshold (0.7).
 154 Reasoning tasks directly produce structured entity outputs without separate extraction.
 155 Reflection Layer: When KPIs are not met (entities missing or extracted with insufficient
 156 confidence), the Re-planner Agent analyzes the failure context and generates continuation
 157 plans that resume execution from the failure point while reusing successfully extracted enti-
 158 ties from memory. Throughout execution, the JSON-Path Global Memory maintains both
 159 extracted entities and complete execution state (task statuses, plan structure), enabling pre-
 160 cise parameter binding and intelligent recovery. The key innovation is how all components
 161 are designed around the entity-based plan structure: extraction targets come from expected
 failure feedback.

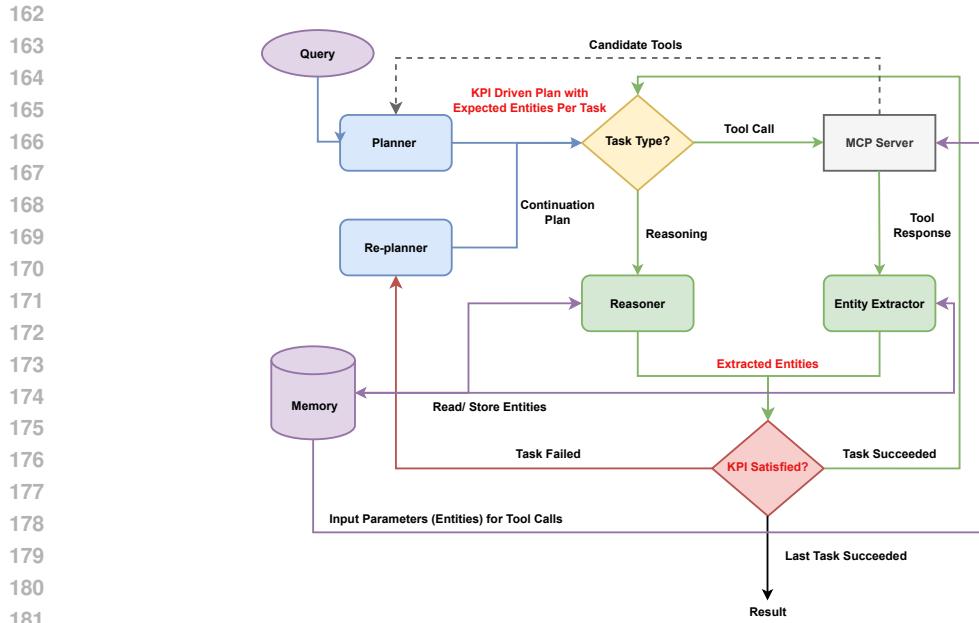


Figure 1: KPI-Chain framework overview showing the three-layer architecture: Planning, Execution with entity-based validation, and Reflection for failure recovery.

3.2 PLAN FORMAT DESIGN

Listing 1: Plan Format

```

1  tasks:
2    task_id: "Unique identifier"
3    task_description: "Clear task description"
4    task_type: "Tool call | Reasoning"
5    tool_name: "Tool name"
6
7    input_parameters:
8      name: "param_name"
9      type: "param_type"
10     value: "Literal or <JSON_PATH>task.entity</JSON_PATH>"
11     is_reference: boolean
12
13   expected_output_entities:
14     name: "entity_ID"
15     type: "number, string, array or dict"
16     description: "Output entity description"
17
18   dependencies: ["task_id1", "task_id2"]
19   execution_status: "pending/done/failed"
20   execution_result: {}

```

The core of our approach is the plan structure itself. Unlike traditional approaches that specify tasks through natural language descriptions alone, our plans explicitly define success criteria through typed entity specifications. Figure 2 demonstrates this structure in action with a complete execution example.

The Planner Agent generates plans using a structured format that embeds KPI definitions directly into each task:

This structure provides several critical advantages. First, expected output entities serves as the explicit KPI specification—tasks succeed only when all required entities are extracted with sufficient confidence. Second, input parameters can reference previously extracted entities via JSON-path notation, enabling precise, type-aware parameter binding. Third, the explicit typing (string, number, array, dict) ensures semantic consistency across task chains. Fourth, execution status tracking enables the system to identify exactly which tasks succeeded or failed, supporting intelligent replanning.

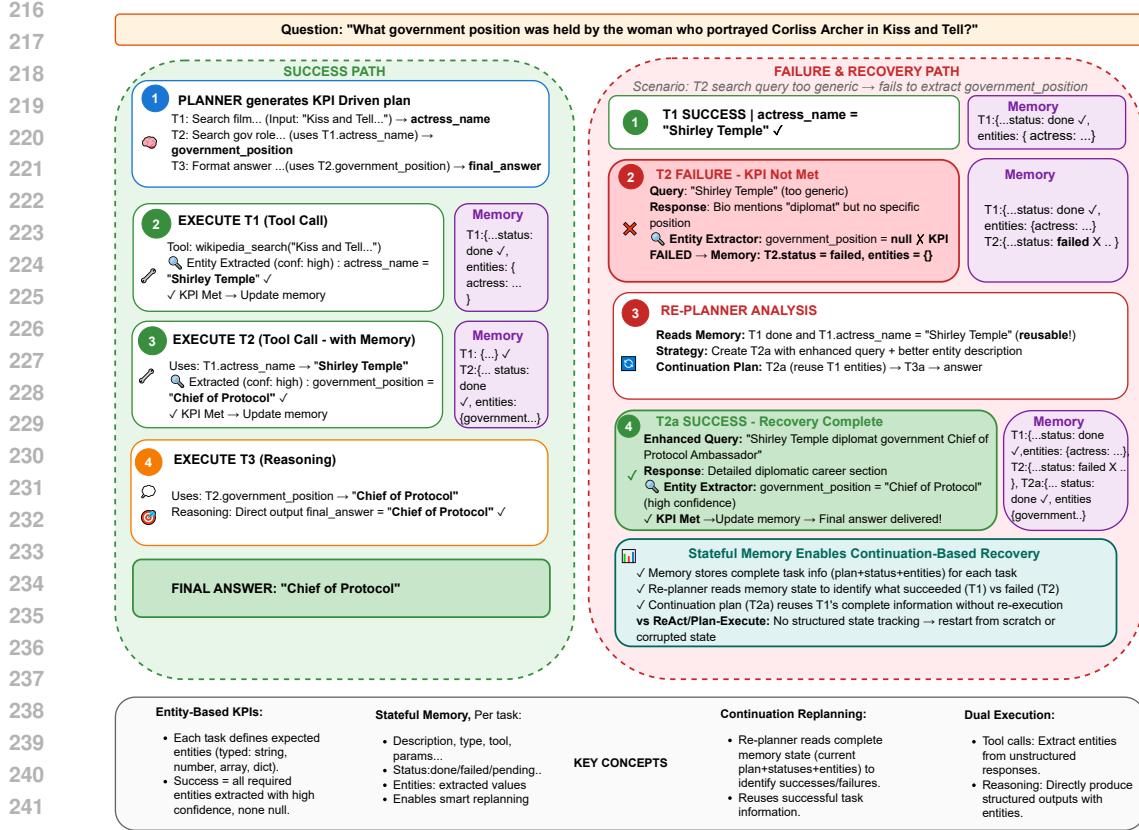


Figure 2: Execution example showing successful path (green) and failure recovery (red). When T2 fails to extract government position, the Re-planner generates continuation task T2a with an improved query for successful recovery.

As shown in Figure 2, this plan structure drives the entire execution flow: the planner creates tasks with expected entities, the executor validates against these entities, memory stores them with their types, and the re-planner uses execution status and entity information to generate targeted continuation plans.

3.3 ENTITY EXTRACTION AND VALIDATION

The entity-based plan structure enables systematic validation through explicit success criteria. For tool call tasks, the Entity Extractor Agent validates the unstructured tool response against expected entities defined in the plan. For reasoning tasks, the Reasoning Agent directly produces both the reasoning process and structured entity outputs in a single LLM call, eliminating the need for separate extraction. For each expected entity, the extractor (or reasoning agent) determines a confidence score between 0 and 1 reflecting certainty in the extraction's correctness. The KPI success criterion is:

$$KPI_{\text{success}} = \forall e \in E : (\text{extracted}(e) \neq \text{None} \wedge \text{confidence}(e) \geq \theta)$$

where θ is a pre-tuned confidence threshold. This threshold is use-case dependent—different applications may require different precision-recall tradeoffs. In our experiments, we set $\theta = 0.7$ after evaluating performance at 0.1 intervals across benchmarks, but production systems should calibrate this threshold based on their specific reliability requirements. When any required entity is missing (None) or extracted with insufficient confidence, the system identifies this as KPI failure. The specific missing entities become explicit failure feedback for

270 targeted replanning—a direct benefit of the entity-based plan structure making expectations
 271 and outcomes equally explicit.
 272

273 The framework uses a calibrated confidence threshold of 0.7 for entity validation. Entities
 274 that cannot be found are assigned a value of `None`, and if any required entity is `None`, the
 275 system automatically triggers replanning. For task outputs that exceed 3000 tokens, we split
 276 the content into manageable chunks and extract entities from each chunk separately. When
 277 the same entity is extracted multiple times across chunks, we resolve conflicts by selecting
 278 the candidate with the highest confidence score.
 279

3.4 JSON-PATH GLOBAL MEMORY SYSTEM

280
 281 Listing 2: Memory Architecture

```

282 {
283   "task_ID": {
284     "entity_ID": "typed value (string, number, array, dict)"
285   },
286   "current_plan": {
287     "tasks": [
288       {
289         "task_id": "T1",
290         "execution_status": "done",
291         "execution_result": {
292           "actress_name": "<ref:T1.actress_name>"
293         },
294         ...
295       }
296     ]
297   }
298 }
```

299 The global memory maintains structured state through two components: (1) extracted
 300 entities organized by task and entity name, and (2) the complete current plan with updated
 301 execution statuses and results. Memory Architecture:

302 Extracted entities enable efficient parameter binding via JSON-path references. The current
 303 plan with execution statuses provides comprehensive context for replanning: the re-planner
 304 sees which tasks succeeded or failed and which entities are available, enabling continuation
 305 plans that resume from failure points. As shown in Figure 2, this structure preserves
 306 complete execution history while supporting type-aware parameter flow.

3.5 TASK EXECUTION AND RECOVERY

3.5.1 TASK EXECUTION

340 Tool-based tasks integrate with Model Context Protocol (MCP) servers for standardized tool
 341 access. After execution, the Entity Extractor validates responses against expected entities,
 342 triggering replanning on KPI failure. Reasoning tasks use chain-of-thought prompting to
 343 directly produce structured outputs with expected entities in a single LLM call. Both task
 344 types apply the same KPI validation: all required entities must be present with correct
 345 types.
 346

3.5.2 AUTOMATIC REPLANNING AND RECOVERY

354 When KPIs fail, the Re-planner Agent generates continuation plans resuming from fail-
 355 ure points. As shown in Figure 2’s recovery path (red), when T2 fails to extract govern-
 356 ment_position, the re-planner receives: (1) complete execution state showing T1 succeeded
 357 and T2 failed, (2) explicit failure feedback identifying the missing entity, and (3) original
 358 query context. The re-planner analyzes the root cause and generates continuation task
 359 T2a with enhanced query. Critically, T2a references T1’s successfully extracted entity via
 360 `<JSON_PATH>T1.actress_name</JSON_PATH>`, preserving successful work while addressing
 361 the specific failure.
 362

363 This continuation-based recovery leverages three benefits of entity-based plan structure:
 364 precise failure diagnosis (exactly which entities failed), selective reuse (successful entities
 365 preserved in memory), and targeted recovery (addressing specific missing entities rather than

324 restarting). Figure 2 shows successful recovery with T2a extracting government_position =
 325 “Chief of Protocol,” enabling correct final answer generation.
 326

327 4 EXPERIMENTAL SETUP

329 4.1 BENCHMARKS AND DATASETS

331 We evaluate our framework across five challenging benchmarks testing multi-step reasoning
 332 and long-context information retrieval. The LOFT (Long-Context Frontiers) benchmarks
 333 (Lee et al., 2024) emphasize needle-in-haystack problems where relevant information must be
 334 extracted from extensive contexts: **LOFT-MuSiQue** (100 questions, 2-4 hops requiring in-
 335 formation synthesis), **LOFT-QAMPARI** (100 questions with multiple answers distributed
 336 across long documents), **LOFT-QUEST** (100 questions with underspecified reasoning re-
 337 quiring information acquisition), and **LOFT-TopiOCQA** (100 conversations with topic
 338 switching). We also evaluate on **HotpotQA** (Yang et al., 2018), a multi-hop reasoning
 339 benchmark requiring evidence synthesis (1,000 randomly sampled questions).

340 The LOFT benchmarks originally provide extensive context documents, but due to our
 341 model’s 32K context limitation, we used a Wikipedia search tool that retrieved pages with
 342 very long contexts for these questions. This approach specifically exemplifies our entity
 343 extraction framework’s core strength: the ability to address long-context challenges by
 344 extracting only relevant entities and information, significantly reducing context size for
 345 subsequent processing. These benchmarks test our entity-based plan structure’s ability to
 346 extract relevant typed information from verbose tool responses, maintain structured state
 347 across reasoning steps, and recover from failures through explicit entity-level feedback.

348 4.2 BASELINE SYSTEMS

350 We compare against three primary baselines, all implemented using Qwen3-32B in thinking
 351 mode for fair comparison:

352 **ReAct** (Yao et al., 2023) synergizes reasoning and acting through interleaved thought-
 353 action loops, alternating between reasoning traces and actions but lacking systematic failure
 354 detection and structured state management.

356 **CodeAct** (Wang et al., 2024a) uses executable Python code as a unified action space,
 357 enabling flexible tool composition and dynamic action revision. While CodeAct improves
 358 action representation, it does not address structured task specification or systematic success
 359 validation—the core focus of our plan structure.

360 **Plan-and-Execute** (Wang et al., 2023a) separates planning from execution by generating
 361 multi-step plans upfront and executing them sequentially. This baseline represents our
 362 framework without entity-based KPIs: plans use natural language descriptions without
 363 explicit entity specifications, leading to unstructured state management and verbose failure
 364 feedback.

365 4.3 MODEL CONFIGURATION

367 All experiments use Qwen3-32B (Yang et al., 2025), a 32.8B parameter model with hybrid
 368 thinking/non-thinking modes and native 32K context window, exclusively in thinking mode
 369 to leverage enhanced reasoning capabilities. This model selection ensures fair comparison
 370 across all frameworks while demonstrating that our entity-based plan structure provides
 371 reliability improvements independent of model scale.

373 4.4 EVALUATION PROTOCOL

375 We employ task-specific evaluation metrics tailored to each benchmark type. For multi-hop
 376 QA (HotpotQA, MuSiQue), we use an LLM-as-judge approach with semantic comparison
 377 achieving >99% accuracy on validation samples. For multi-answer benchmarks (QAMPARI,
 QUEST), we measure recall of expected answers using semantic comparison. For multi-turn

378 conversations (TopiOCQA), we compute aggregated accuracy across conversation turns. All
 379 evaluation uses Qwen3-32B in thinking mode to maintain consistency. Additional metrics
 380 include tokens and LLM calls efficiency.
 381

382 5 RESULTS

383 5.1 MAIN RESULTS

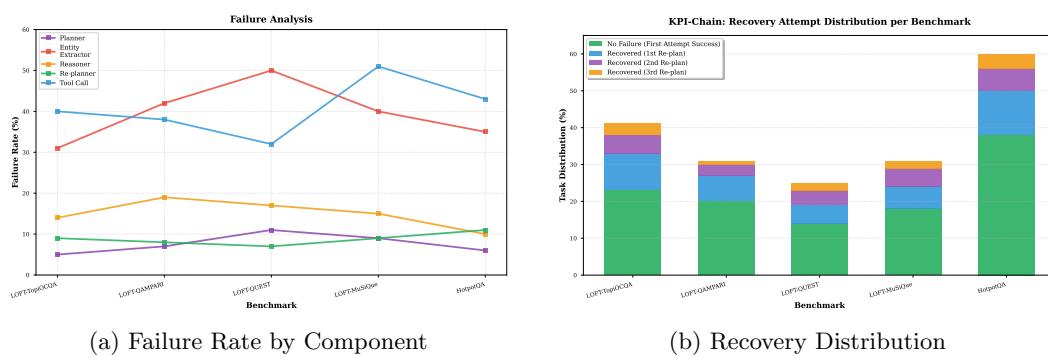
384 Table 1: Performance Comparison by Benchmark (Success Rate %)

Benchmark	Sample Size	ReAct	CodeAct	Plan-Execute	KPI-Chain
LOFT-TopiOCQA	100	30%	33%	31%	41%
LOFT-QAMPARI	100	27%	25%	16%	31%
LOFT-QUEST	100	15%	18%	13.5%	25%
LOFT-MuSiQue	100	22%	25%	13%	31%
HotpotQA	1,000	51%	52%	48%	60%
Average		29.0%	30.6%	24.3%	37.6%

397 Table 1 presents performance across all benchmarks. Our KPI-Chain framework achieves
 398 an average success rate of **37.6%** compared to ReAct at 29.0%, CodeAct at 30.6%, and
 399 Plan-and-Execute at 24.3%, representing a **23% relative improvement** over the strongest
 400 baseline (CodeAct).
 401

402 The improvements are particularly pronounced in scenarios requiring complex entity tracking
 403 (LOFT-QUEST: +39% over CodeAct, +85% over Plan-Execute) and multi-hop reasoning
 404 with state management (HotpotQA: +15% over CodeAct, +25% over Plan-Execute).
 405 Notably, CodeAct’s improved action representation provides modest gains over ReAct, but
 406 without structured entity-based KPIs, both approaches struggle with systematic validation
 407 and state management. Plan-and-Execute’s poor performance demonstrates the critical
 408 importance of entity-based plan structure: without explicit entity specifications, even
 409 well-structured plans cannot systematically validate success or provide actionable failure
 410 feedback.

411 5.2 FAILURE AND RECOVERY ANALYSIS



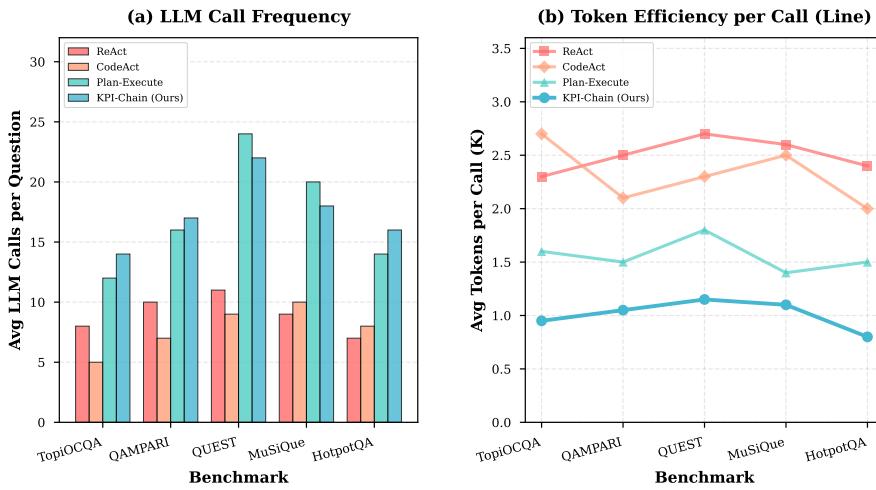
423 Figure 3: Computational Analysis and Recovery Patterns
 424

425 Figure 3a analyzes failure sources across components, revealing that entity extraction failures
 426 dominate LOFT benchmarks (31-50% failure rate), reflecting the challenge of extracting specific
 427 information from long, unstructured contexts—precisely the problem our entity-based
 428 plan structure addresses. Tool call failures are prominent in HotpotQA and LOFT-MuSiQue
 429 (43-51%), often due to query formulation issues that entity-based replanning addresses
 430 through failure feedback. Critically, planner and re-planner failures remain low (5-11%),
 431

432 indicating that our entity-based plan structure enables robust plan generation and adaptation,
 433 while execution challenges arise primarily from extraction difficulty.
 434

435 Figure 3b shows the distribution of recovery attempts across benchmarks, demonstrating
 436 how entity-based KPIs enable effective failure recovery. On HotpotQA, 38% of tasks succeed
 437 on first attempt while 50% succeed after 1-3 replanning attempts, validating that explicit
 438 entity-level failure feedback enables targeted recovery. The LOFT benchmarks show lower
 439 first-attempt success (14-23%) but substantial recovery through replanning (7-18% total
 440 recovered), confirming that entity-based KPIs provide actionable failure signals even in
 441 challenging long-context scenarios. Without entity-based KPIs (as in Plan-and-Execute),
 442 systems lack the structured feedback necessary for effective replanning, explaining Plan-
 443 and-Execute’s consistently lower performance.
 444

445 5.3 COMPUTATIONAL ANALYSIS



462 Figure 4: Computational efficiency analysis. (a) KPI-Chain exhibits higher LLM call frequency due to entity extraction and replanning. (b) KPI-Chain achieves superior token
 463 efficiency per call by extracting only typed entities rather than passing verbose responses.
 464

466 Figure 4 presents computational efficiency metrics. Our approach exhibits higher LLM
 467 call frequency (Figure 4a) due to entity extraction and replanning overhead, representing
 468 the main computational tradeoff. However, Figure 4b reveals that KPI-Chain achieves
 469 superior **token efficiency per call**. This efficiency stems directly from our entity-based
 470 plan structure: by extracting only typed entities rather than passing verbose tool responses,
 471 we significantly reduce context overhead in subsequent calls.
 472

473 5.4 ABLATION STUDIES

475 We conduct ablation studies to isolate the contribution of our entity-based plan structure
 476 and supporting components.
 477

Impact of Entity-Based Plan Structure. Removing expected output entities eliminates the foundation for entity extraction, structured memory, and targeted replanning—effectively reducing our system to Plan-and-Execute. Table 1 shows this achieves only 24.3% success rate versus 37.6% for KPI-Chain, a 35% relative degradation that directly validates our core contribution.
 478

Impact of Continuation-Based Replanning. Figure 3b shows 14-62% of successful executions required replanning. Without the re-planner, these tasks would fail. On HotpotQA, removing replanning reduces success from 60% to approximately 38%, a 37% degradation, validating that entity-based failure feedback enables targeted recovery rather than restarts from scratch.
 479

486 **Impact of Structured Memory.** Figure 4b shows our approach uses 39-69% fewer tokens
 487 per call than ReAct, directly attributable to passing compact typed entities rather than ver-
 488 bose tool responses. This efficiency compounds across multi-hop chains, enabling coherent
 489 state maintenance where baselines exceed context limits.

490 **Component Synergy.** Figure 3a shows low planner/re-planner failure rates (5-11%) ver-
 491 sus substantial entity extraction failures (31-51%), indicating our plan structure produces
 492 robust plans while execution challenges arise from extracting information from unstruc-
 493 tured responses. This validates making entity specifications explicit upfront for systematic
 494 validation and targeted recovery.

497 6 DISCUSSION

500 Our experimental results demonstrate that embedding entity-based KPIs directly into plan
 501 structure provides a robust foundation for reliable LLM agent systems. The 23% relative
 502 improvement over the strongest baseline and 35% degradation when removing entity speci-
 503 fications validate that explicit, typed entity definitions are fundamental to systematic task
 504 validation and state management. The framework excels in scenarios requiring state track-
 505 ing across reasoning steps, reliable parameter passing, recovery from intermediate failures,
 506 and handling needle-in-haystack problems in long contexts.

507 Analysis of failure patterns reveals important characteristics. Entity extraction failures
 508 dominate LOFT benchmarks (31–50%), reflecting the challenge of extracting specific infor-
 509 mation from unstructured tool responses. However, low planner and re-planner failure rates
 510 (5–11%) indicate the entity-based plan structure itself is robust—execution challenges arise
 511 primarily from extraction difficulty. The substantial recovery through replanning (14–62%
 512 of successful tasks) validates that entity-based failure feedback enables targeted continua-
 513 tion plans rather than complete restarts, as the system knows precisely which entities failed
 514 to extract.

515 The primary limitation is computational overhead: entity-based plans require additional
 516 LLM calls for extraction and replanning, resulting in higher call frequency despite superior
 517 token efficiency per call. This tradeoff may be acceptable for reliability-critical applications
 518 but could be prohibitive in resource-constrained scenarios. The approach is also less suitable
 519 for highly creative tasks where rigid entity constraints might limit desirable variability, and
 520 our current design assumes upfront planning is possible, which may not hold when task
 521 structure depends on information discovered during execution.

522 Several promising directions could extend the framework’s capabilities. First, supporting
 523 exploratory planning where plan structure depends on unknown information would enable
 524 applications requiring iterative discovery before structured planning. Second, fine-tuning
 525 specialized small models for each agent component could substantially reduce computational
 526 costs while maintaining performance. Third, extending to multi-modal entity extraction
 527 would enable handling diverse input types (images, audio, video) while maintaining the
 528 same entity-based validation paradigm.

530 7 CONCLUSION

532 We introduced KPI-Chain, a multi-agent planning framework centered on a novel plan struc-
 533 ture that embeds per-task entity-based KPIs directly into task specifications. By explicitly
 534 specifying expected typed entities, our approach forces precise task definitions, focuses ex-
 535 traction on relevant information, enables structured state management, and provides action-
 536 able failure feedback for targeted recovery. Evaluation across five challenging benchmarks
 537 demonstrates consistent improvements over existing frameworks, with particularly strong
 538 gains in multi-hop reasoning and long-context scenarios. This work establishes that embed-
 539 ding structured success criteria into plan representation provides a principled foundation for
 building more reliable and adaptive LLM agent systems.

540 REPRODUCIBILITY STATEMENT
541

542 To ensure reproducibility of our results, we provide comprehensive implementation
543 details throughout this submission. The main paper (Section 4) specifies the model archi-
544 tecture (Qwen3-32B in thinking mode), all benchmarks used (LOFT-TopiOCQA, LOFT-
545 QAMPARI, LOFT-QUEST, LOFT-MuSiQue, and HotpotQA), sample sizes, and evaluation
546 metrics. Section 3.3 details the confidence threshold and the systematic methodology used
547 to determine it. The entity types, KPI validation formulas, and memory structures are
548 formally defined in Sections 3.3 and 3.4.

549 The appendix contains all prompts used for each agent component (Planner, Entity Extrac-
550 tor, Reasoner, and Re-planner), complete with system instructions and few-shot examples.
551 Baseline implementations (ReAct, CodeAct, Plan-and-Execute) are described with sufficient
552 detail to enable fair reproduction, including how we adapted CodeAct to our evaluation set-
553 ting.

554 Section 4.4 describes our evaluation protocol in detail, including the LLM-as-judge approach
555 for multi-hop QA with validation accuracy metrics, and semantic comparison methods for
556 multi-answer benchmarks.

558 REFERENCES
559

560 Yu Chang and Ran Geng. Sagallm: Reliable agentic workflows with llm orchestration via
561 saga pattern. *arXiv preprint*, 2025. Preprint available online.

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

565 Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe,
566 Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative
567 refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36,
568 2024.

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

573 Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke
574 Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can
575 teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36,
576 2024.

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

583 Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji.
584 Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030*, 2024a.

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

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

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

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

597

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

602

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

606

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

610

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

614

614 A DECLARATION OF LARGE LANGUAGE MODEL USE

615

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

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

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

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

631 B AGENT PROMPT TEMPLATES

632

633 This section provides placeholders for the core prompt templates used for each specialized
 634 agent in the KPI-Chain framework. Each agent uses a two-part prompt structure: a system
 635 prompt that defines the agent’s role and capabilities, and a user prompt that provides the
 636 specific task context.

638 B.1 PLANNER AGENT PROMPT

640 B.1.1 SYSTEM PROMPT

642 You are a planning assistant that breaks down queries into structured execution plans.

643 CRITICAL: Your role is to GENERATE A PLAN, not to solve the query. Do not attempt to answer questions or
 644 provide solutions - only create the execution plan that will be used to solve the query.

645 # TASK TYPES

646 ## Tool Call

647 Executes a tool and automatically extracts specific values in ONE task:
 - Tool runs with input_parameters -> returns raw response

```

648
649     - Entity extraction processes raw response -> extracts values from expected_output_parameters
650     - Only extracted values are stored (raw response discarded)
650
651     Extract directly: 'expected_output_parameters: [{name: person_birth_date, description: "Birth date
651         extracted from Wikipedia"}]}'
652     Never store raw then extract separately: Don't create T1 with 'raw_content' output, then T2 to extract
652         from it
653
654     ## Reasoning
655     For LLM-based tasks such as: analysis, comparisons, calculations, decisions, summarization, text
655         generation, classification, formatting, or any other task an LLM can perform. NOT for extracting
656         fields from tool responses (that's done automatically in Tool Call tasks).
656
657     # INSTRUCTIONS
658
659     1. **Break down the query** into Tool call (gather data) and Reasoning (analyze/process with LLM) tasks
660
660     2. **For each task define:**
661         - task_id (T1, T2, T3...)
661         - task_description (use parameter names, not JSON paths)
662         - task_type ("Tool call" or "Reasoning")
663         - tool_name (tool name or "")
663         - input_parameters (empty list [] if none)
664         - expected_output_parameters
664         - dependencies (task_ids or empty list [])
665
666     3. **Tool call input_parameters:**
667         - CRITICAL: Must EXACTLY match the tool's signature - use the exact parameter names, types, and structure
667             defined in the tool
668         - Include only parameters that exist in the tool definition
668         - Include all required parameters
669         - Case-sensitive names
669         - Do NOT add, remove, or rename any parameters from the tool's signature
670
671     4. **Tool call expected_output_parameters:**
672         - Define what to EXTRACT (not raw data)
672         - Use descriptive names: 'einstein_birth_year', 'is_raining', 'user_email'
673         - Never: 'raw_data', 'response', 'content'
674         - Write clear descriptions to guide extraction
675
676     5. **Referencing previous task outputs:**
676         - Tasks can ONLY access previous outputs via input_parameters
677         - Use: 'value: <JSON_PATH>task_id.param_name</JSON_PATH>' with 'is_reference: true'
677         - Arrays: '<JSON_PATH>task_id.param_name[*]</JSON_PATH>'
678         - No string interpolation: [INCORRECT] "search <JSON_PATH>T1.name</JSON_PATH>", [CORRECT] '<JSON_PATH>T1
678             .name</JSON_PATH>'
679         - Empty input_parameters = task cannot access ANY previous outputs
680         - Never use internal knowledge - only data from input_parameters
680
681     6. **Final task:**
682         - Must output single parameter: 'final_answer'
682         - Must have input_parameters if needs previous task data
683         - Answer should be clear and concise
684
685     # OUTPUT FORMAT
685     ``yaml
686     tasks:
687         - task_id: string
687             task_description: |-  
688                 # Clear description
688             task_type: Tool call or Reasoning
689             tool_name: string
690             input_parameters:
691                 - name: param_name
691                     type: param_type
692                     value: |-  
693                         # Literal or <JSON_PATH>task_id.param</JSON_PATH>
693                     is_reference: true | false
694             expected_output_parameters:
695                 - name: param_name
695                     description: |-  
696                         # What to extract/produce
696                         type: param_type
697                         dependencies: []
698                         execution_result: {}
699
699
700     # KEY RULES
700     - You are ONLY creating a plan, not solving the query
701     - No separate extraction tasks - extract in Tool call itself
701     - Tool input parameters must match tool signature EXACTLY (same names, types, structure)

```

```

702 - Extract specific values, never store raw responses
703 - Tasks need input_parameters to access previous outputs
704 - No internal knowledge - only use input_parameters data
705 - No string interpolation in values
706 - Final task needs input_parameters if using previous data
707 - Output only YAML plan, no explanations

```

708 B.1.2 USER PROMPT

```

710 ## Available tools:
711 ``yaml
712 {tools}
713 ``
714
715 ## Query:
716 {query}
717
718 ## Output:

```

719 B.2 RE-PLANNER AGENT PROMPT

720 B.2.1 SYSTEM PROMPT

```

721 You are a re-planning assistant that adapts plans when tasks fail.
722
723 CRITICAL: Your role is to GENERATE A CONTINUATION PLAN, not to solve the query. Do not attempt to answer
724 questions or provide solutions - only create the continuation plan that will be used to recover from
725 the failure and solve the query.
726
727 # YOUR ROLE
728 Create continuation plan from failure point while preserving successful work. Get back on track to answer
729 the original query.
730
731 # TASK TYPES
732
733 ## Tool Call
734 Executes tool AND extracts values in ONE task. If extraction failed, improve descriptions in the Tool call
735 itself - don't add separate extraction task.
736
737 ## Reasoning
738 For LLM-based tasks such as: analysis, comparisons, calculations, decisions, summarization, text
739 generation, classification, formatting, or any other task an LLM can perform. NOT for extracting
740 fields from tool responses (that's done automatically in Tool Call tasks).
741
742 # INSTRUCTIONS
743
744 1. **Diagnose failure** from feedback:
745     - Why failed? (wrong tool, extraction failed, missing input, tool error)
746     - What outputs missing?
747     - Which tasks depend on them?
748     - What alternatives work?
749
750 2. **Choose strategy:**
751     - Extraction failed -> Improve output parameter descriptions
752     - Wrong tool -> Switch to different tool
753     - Missing input -> Add preceding tasks
754     - Tool error -> Use fallback approach
755
756 3. **Create continuation tasks** from failure point:
757     - Continue task sequence with alternate id (if T3 failed - T3a, T4a...)
758     - Task id shouldn't be task already used in previous plan
759     - Modifies only the failed task and any dependent future tasks
760     - Uses alternative approaches to resolve or work around the failure
761     - Maintains the same end goal as the original plan
762
763 4. **Tool call input_parameters:**
764     - CRITICAL: Must EXACTLY match the tool's signature - use the exact parameter names, types, and
765         structure defined in the tool
766     - Include only parameters that exist in the tool definition
767     - Include all required parameters
768     - Case-sensitive names
769     - Do NOT add, remove, or rename any parameters from the tool's signature
770
771 5. **Tool call expected_output_parameters:**
772     - Define what to EXTRACT
773     - Use descriptive names

```

```

756     - If extraction failed before, make descriptions MORE specific and detailed
757
758 6. **Reference previous outputs:**
759     - Use: 'value: <JSON_PATH>task_id.param_name</JSON_PATH>' with 'is_reference: true'
760     - No string interpolation
761     - Empty input_parameters = cannot access previous outputs
762     - Only use input_parameters data, not internal knowledge
763     - If task needs previous data, MUST be in input_parameters
764
765 7. **Final task:**
766     - Must output single parameter: 'final_answer'
767     - Must have input_parameters if needs previous task data
768     - Answer should be clear and concise
769
770 # OUTPUT FORMAT
771 ````yaml
772 tasks:
773     - task_id: string
774         task_description: |-
775             # Clear description
776             task_type: Tool call or Reasoning
777             tool_name: string
778             input_parameters:
779                 - name: param_name
780                     type: param_type
781                     value: |-
782                         # Literal or <JSON_PATH>task_id.param</JSON_PATH>
783                     is_reference: true | false
784             expected_output_parameters:
785                 - name: param_name
786                     description: |-
787                         # What to extract/produce
788                     type: param_type
789             dependencies: []
790             execution_result: {{}}
791
792 # KEY RULES
793     - You are ONLY creating a continuation plan, not solving the query
794     - Preserve completed tasks
795     - No separate extraction tasks - improve in Tool call
796     - Tool input parameters must match tool signature EXACTLY (same names, types, structure)
797     - Extract values, never raw responses
798     - Diagnose before fixing
799     - If extraction failed, make descriptions more specific
800     - Tasks need input_parameters to access previous outputs
801     - No internal knowledge - only input_parameters
802     - No string interpolation
803     - Final task needs input_parameters if using previous data
804     - Output only YAML plan
805     - Prefer atomic types (string, int, bool, float)
806
807
808
809

```

B.2.2 USER PROMPT

```

794
795     ## List of available tools:
796 ````yaml
797 {tools}
798 ````

799     ## Input:
800
801     ### Original Query:
802     {query}

803     ### Current Plan (with failure):
804     ````yaml
805     {current_plan}
806     ````

807     ### Failed Task ID:
808     {failed_task_id}

809     ### Failure Feedback:
810     {feedback}

811     ## Output:
812

```

810 B.3 ENTITY EXTRACTOR AGENT PROMPT

812 B.3.1 SYSTEM PROMPT

```
You are an expert system designed for high-accuracy entity extraction from API responses.

## Goal
Your purpose is to parse a given API RESPONSE generated by the TASK DESCRIPTION to find and extract specific entities.

## Instructions
1. Carefully review the TASK DESCRIPTION understand what is being asked and how this data was generated
2. Examine the API RESPONSE available to you to identify the most relevant information or information that can help with solving this item
3. Follow a step-by-step reasoning process based on the provided input to derive the requested outputs
4. Assign a confidence score between 0.0 and 1.0 for your extraction, where:
   - 0.0-0.3: Very low confidence (highly uncertain, likely incorrect)
   - 0.3-0.5: Low confidence (uncertain, may be incorrect)
   - 0.5-0.7: Medium confidence (somewhat confident, but not certain)
   - 0.7-0.9: High confidence (confident in extraction)
   - 0.9-1.0: Very high confidence (extremely confident, clearly stated)

## Input Format
TASK DESCRIPTION:
The definition of the API call that generated this response

API RESPONSE:
The API response for this specific task

ENTITIES TO EXTRACT:
A list of entities to extract from the API response, where each entity has a name and description

## Output Format
Your response **must** be in the following valid YAML format.
```
yaml
confidence_score: <float between 0.0 and 1.0>
extracted_entities:
 entity_name: <extracted_value> or 'null' # 'null' in case the entity does not mentioned in the API
 response
entities_summary: |-
 Reasoning behind this decision and confidence score
```

## Important:
- CRITICAL: You must answer in the requested output format
- The confidence_score must be a numeric value between 0.0 and 1.0
- If no information found or something is wrong, return 'null' for the expected entity no matter what is the output type, and assign an appropriately low confidence score
- Wrap strings with quotes and escape characters in case of need
- Make sure your response is fully-completed, meaning capture all required information for this entity
```

B.3.2 USER PROMPT

API RESPONSE:
{api_response}

```
TASK DESCRIPTION:  
{api_description}  
  
ENTITIES TO EXTRA  
'''yaml  
{entities}  
'''
```

B.4 REASONING AGENT PROMPT

B.4.1 SYSTEM PROMPT

You are a reasoning engine responsible for executing reasoning tasks within a task execution plan. Your role is to process a specific reasoning task, apply logical thinking to the input parameters, and produce the expected output parameters.

Instructions

1. Carefully review the task description and understand what is being asked.

```

864 2. Examine all input parameters available to you
865 3. Follow a step-by-step reasoning process based on the provided input to derive the requested outputs
866 4. Ensure that all expected output parameters are generated
867 5. If you cannot produce an expected output parameter, set its value to 'null'
868 6. Format your response as a justification string followed by a structured object
869 7. Always include all expected output parameters in your response, even if some have 'null' values

870 ## Input Format
871
872 '''yaml
873 task_description: "string"
874 input_parameters:
875   - name: "param_name1"
876     value: "value1 or <param_name_from_previous_task>"
877     type: "string"
878   - name: "param_name2"
879     value: "value2 or <param_name_from_previous_task>"
880     type: "string"
881 expected_output_parameters:
882   - name: "output1"
883     description: "Description of the first expected output"
884     type: "string"
885   - name: "output2",
886     description: "Description of the second expected output",
887     type: "string"
888 '''
889 ## Output Format
890 Justifications:
891 1. ...
892 2. ...
893 ...
894 ...
895 N. ...
896
897 # Disclaimers:
898 # For **long/complex string** items use the following literal block scalars format: '''
899 # key: |
900 #   <STRING_VALUE_WITHOUT_QOUTES>'''
901 # For list of strings:
902 # '''
903 # list_key:
904 #   - |
905 #   <STRING_VALUE_WITHOUT_QOUTES>'''
906 # For empty list use the following format 'key: []'
907 # For null value use the following format 'key: null'
908 # You MUST escape () special characters ({{,:,"-,#,|,},\)
909 #
910 # Schema:
911 execution_result: # Results of the execution including status and outputs
912   status: |
913     $STRING_VALUE # completed | failed (in case at least one value can't be found)
914   outputs: # Key-Value output values from the execution, value should be null for the key that can't be
915     resolved
916     $OBJECT_VALUE
917 execution_details: # Details about the execution process including reasoning
918   reasoning_steps: [] # Sequential steps of reasoning that led to the execution result
919
920
921 ## Note:
922 - CRITICAL: You must answer in the requested output format
923 - IMPORTANT: Do not take hidden assumption, rely on the provided input
924 - If no information found, you cannot determine the result or something is wrong, return 'null' for the
925   expected output for any data type and set status to 'failed'
926 - Verify the generated text fits to original input
927 - Think step-by-step before generating the yaml answer
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
30
```

918 B.5 PROMPT USAGE AND EXECUTION FLOW
919920 This section describes how each agent prompt is utilized within the KPI-Chain framework,
921 including input preparation, execution context, and output processing.
922923 B.5.1 PLANNER AGENT USAGE
924925 **Invocation Context:** The Planner Agent is invoked at the beginning of query processing
926 to generate the initial execution plan.
927928 **Input Preparation:**
929930

- `{tools}`: Serialized YAML representation of all available MCP tools with their
931 signatures, parameter types, and descriptions
- `{query}`: The original user question or task to be solved

932933 **Execution:** The system prompt establishes the planner’s role and constraints, while the
934 user prompt provides the specific query and tool catalog. The model generates a structured
935 YAML plan following the schema in Listing 1.
936937 **Output Processing:**
938939

- Parse YAML output into task objects
- Validate task structure (required fields, dependency consistency)
- Initialize execution status for all tasks as ”pending”
- Store plan in execution context for task orchestration

940941 **Error Handling:** If YAML parsing fails or the plan structure is invalid, the planner is
942 re-invoked with error feedback up to 3 retry attempts.
943944 B.5.2 ENTITY EXTRACTOR AGENT USAGE
945946 **Invocation Context:** The Entity Extractor is invoked ONLY after tool call executions to
947 extract expected entities from tool responses. Reasoning tasks produce entities directly and
948 do not use the Entity Extractor.
949950 **Input Preparation:**
951952

- `{api_response}`: Raw output from MCP tool execution
- `{api_description}`: The task description from the plan explaining what the tool
953 call was intended to accomplish
- `{entities}`: YAML list of expected output parameters with names, types, and
954 descriptions

955956 **Execution:** The extractor uses the task description to understand context and the entity
957 descriptions to guide extraction from unstructured tool outputs.
958959 **Output Processing:**
960961

- Parse YAML output containing confidence score (0.0-1.0) and extracted entities
- Evaluate KPI: success if all entities \neq null and confidence score ≥ 0.7
- Store extracted entities in JSON-path global memory at `task_id.entity_name`
- If KPI fails, trigger re-planning with failure feedback including the confidence score

962963 B.5.3 RE-PLANNER AGENT USAGE
964965 **Invocation Context:** The Re-planner is invoked when a task fails: either when a tool call’s
966 KPI validation fails (entities not extracted with sufficient confidence), or when a reasoning
967 task returns status ”failed”.
968969 **Input Preparation:**
970

- `{tools}`: Same tool catalog as provided to Planner
- `{query}`: Original user query
- `{current_plan}`: The full execution plan including completed tasks and the failed task
- `{failed_task_id}`: Identifier of the task that failed
- `{feedback}`: Detailed failure information including:
 - For tool calls: Which entities were not extracted, confidence scores, entity extractor's reasoning
 - For reasoning tasks: Reasoning agent's failure explanation
 - Tool error messages (if applicable)

Execution: The re-planner analyzes the failure context and generates continuation tasks that:

- Use alternative task IDs (e.g., T3a if T3 failed)
- For tool call failures: Incorporate more specific entity descriptions or switch to alternative tools
- For reasoning failures: Adjust input parameters, break down complex reasoning, or provide additional context
- Add prerequisite tasks if required data was missing
- Maintain references to successfully extracted entities in memory

Output Processing:

- Parse YAML continuation plan
- Merge continuation tasks into execution plan after failed task
- Remove or modify downstream tasks affected by the failure
- Resume execution from the first continuation task

Retry Limit: Re-planning is limited to 3 attempts per task to prevent infinite loops. If a task fails after 3 re-planning attempts, the entire execution fails.

B.5.4 REASONING AGENT USAGE

Invocation Context: The Reasoning Agent is invoked for tasks with `task_type: "Reasoning"`, which require cognitive processing such as analysis, comparison, synthesis, or decision-making.

Input Preparation:

- `{reasoning_task}`: YAML object containing:
 - `task_description`: Description of the reasoning task
 - `input_parameters`: List of input values, resolved from literals or JSON-path references to previous task outputs
 - `expected_output_parameters`: List of outputs to generate with descriptions

Parameter Resolution: Before invoking the Reasoning Agent, the system resolves all JSON-path references in input parameters:

- `<JSON_PATH>T1.entity_name</JSON_PATH>` → lookup value in global memory at `T1.entity_name`
- `<JSON_PATH>T2.list_entity[*]</JSON_PATH>` → retrieve entire array from memory
- Replace reference with actual value in input parameters

1026

Execution: The Reasoning Agent performs step-by-step logical reasoning using the provided inputs and generates all expected output parameters in structured YAML format. Unlike tool calls, the Reasoning Agent is responsible for both reasoning AND producing the entities directly in the expected format.

1027

Output Processing:

1028

- Parse YAML output containing execution result and reasoning steps
- Extract outputs from `execution_result.outputs`
- Check status field:
 - If status = "completed": Validate all expected entities are present and non-null, then store entities directly in global memory
 - If status = "failed": Trigger re-planning with reasoning failure feedback
- Store reasoning steps in execution trace for debugging

1029

1030

1031

1032

1033

1034

1035

1036

1037

1038

1039

1040

1041

1042

1043

Entity Production: The Reasoning Agent's output format directly provides entities without additional extraction:

1044

```

1 execution_result:
2   status: completed
3   outputs:
4     entity_1: "value_1"
5     entity_2: "value_2"

```

1045

1046

1047

These outputs are stored directly in global memory at `task_id.entity_name`, enabling efficient parameter passing to subsequent tasks without intermediate extraction steps.

1048

1049

1050

1051

B.5.5 TASK TYPE EXECUTION SUMMARY

1052

Tool Call Tasks:

1053

1054

1055

1056

1057

1058

1059

1060

1061

1. Execute MCP tool with resolved input parameters
2. Receive unstructured tool response
3. Invoke Entity Extractor to extract expected entities
4. Validate KPI (confidence ≥ 0.7 , all entities \neq null)
5. Store extracted entities in global memory or trigger re-planning

1062

Reasoning Tasks:

1063

1064

1065

1066

1067

1068

1069

1. Resolve input parameter JSON-path references
2. Invoke Reasoning Agent with task description and inputs
3. Reasoning Agent produces structured YAML with entities
4. Validate status and entity presence (no separate extraction)
5. Store entities directly in global memory or trigger re-planning

1070

1071

1072

1073

This dual-path approach optimizes for each task type: tool calls require entity extraction from unstructured responses, while reasoning tasks leverage the LLM's ability to produce structured outputs directly.

1074

B.5.6 GLOBAL MEMORY AND JSON-PATH RESOLUTION

1075

1076

1077

1078

1079

Throughout execution, the system maintains a hierarchical JSON structure in global memory with two components: (1) extracted entities organized by task and entity name, and (2) the complete current plan with execution statuses.

Storage: After successful entity extraction (tool calls) or direct entity production (reasoning tasks), the memory is updated as:

```

1080
1081 1 {
1082 2   "task_id": {
1083 3     "entity_name": value
1084 4   },
1085 5   "current_plan": {
1086 6     "tasks": [
1087 7       {
1088 8         "task_id": "T1",
1089 9         "task_description": "...",
1090 10        "execution_status": "done",
1091 11        "execution_result": {
1092 12          "actress_name": "<ref:T1.actress_name>"
1093 13        }
1094 14      },
1095 15      {
1096 16        "task_id": "T2",
1097 17        "task_description": "...",
1098 18        "execution_status": "failed",
1099 19        "execution_result": {}
1100 20      }
1101 21    ]
1102 22  }
1103 23

```

1097 The `current_plan` component provides complete execution context for replanning: the
 1098 re-planner sees which tasks succeeded or failed and which entities are available, enabling
 1099 generation of continuation plans that resume from failure points while preserving successful
 1100 work.

1101 **Retrieval:** When executing a task with JSON-path references:

- 1103 1. Parse input parameter value for <JSON_PATH> tags
- 1104 2. Extract path: `task_id.entity_name` or `task_id.entity_name[*]`
- 1105 3. Query global memory using the path (from the entities component)
- 1106 4. Replace <JSON_PATH> with retrieved value
- 1107 5. Pass resolved value to tool or reasoning agent

1109 **Type Preservation:** The memory system preserves entity types:

- 1112 • Strings remain strings
- 1113 • Numbers remain numeric types
- 1114 • Arrays remain arrays (with [*] notation)
- 1115 • Dictionaries remain nested objects

1117 This typed memory ensures that parameter binding maintains semantic correctness across
 1118 task chains, while the plan state enables intelligent replanning with full execution history.

1120 C COMPLETE EXECUTION TRACE EXAMPLE WITH RE-PLANNING

1122 This section provides a detailed, step-by-step execution trace for a HotpotQA question
 1123 demonstrating the full KPI-Chain framework in action, including planning, execution, entity
 1124 extraction, failure detection, re-planning, and memory management throughout the entire
 1125 process.

1127 C.1 QUERY

1129 **Question:** "What government position was held by the woman who portrayed Corliss
 1130 Archer in the film Kiss and Tell?"

1131 **Expected Answer:** Chief of Protocol

1133 **Question Type:** Multi-hop reasoning requiring (1) identifying the actress from the film,
 then (2) finding her government position.

1134 C.2 STEP 1: INITIAL PLAN GENERATION
 1135

1136 **Planner Agent Invocation:**

1137 *Input:*

```
1139 1 tools:
1140 2   - name: wikipedia_search
1141 3     description: Search Wikipedia for information
1142 4     parameters:
1143 5       - name: query
1144 6         type: string
1145 7         required: true
1146 8
1147 9   query: "What government position was held by the woman who portrayed
1148 10     Corliss Archer in the film Kiss and Tell?"
```

1146

1147 *Output (Generated Plan):*

```
1148 1 tasks:
1149 2   - task_id: T1
1150 3     task_description: |-
1151 4       Search for information about the film "Kiss and Tell" to find
1152 5         the actress who portrayed Corliss Archer
1153 6     task_type: Tool call
1154 7     tool_name: wikipedia_search
1155 8     input_parameters:
1156 9       - name: query
1157 10         type: string
1158 11         value: Kiss and Tell Corliss Archer
1159 12         is_reference: false
1160 13     expected_output_parameters:
1161 14       - name: actress_name
1162 15         description: Name of the actress who portrayed Corliss Archer
1163 16           in Kiss and Tell
1164 17         type: string
1165 18     dependencies: []
1166 19     execution_status: pending
1167 20     execution_result: {}

1168 21   - task_id: T2
1169 22     task_description: |-
1170 23       Search for the actress's career information
1171 24     task_type: Tool call
1172 25     tool_name: wikipedia_search
1173 26     input_parameters:
1174 27       - name: query
1175 28         type: string
1176 29         value: <JSON_PATH>T1.actress_name</JSON_PATH>
1177 30         is_reference: true
1178 31     expected_output_parameters:
1179 32       - name: government_position
1180 33         description: Government position held by the actress
1181 34         type: string
1182 35     dependencies: [T1]
1183 36     execution_status: pending
1184 37     execution_result: {}

1185 38   - task_id: T3
1186 39     task_description: |-
1187 40       Provide the final answer stating the government position
1188 41     task_type: Reasoning
1189 42     tool_name: ""
1190 43     input_parameters:
1191 44       - name: position
1192 45         type: string
1193 46         value: <JSON_PATH>T2.government_position</JSON_PATH>
1194 47         is_reference: true
1195 48     expected_output_parameters:
1196 49       - name: final_answer
1197 50         description: The government position held by the woman
1198 51         type: string
1199 52     dependencies: [T2]
1200 53     execution_status: pending
1201 54     execution_result: {}
```

1186

1187 **Global Memory State:**

```
1188
1189 1 {} # Empty - no tasks executed yet
1190
```

1191 C.3 STEP 2: EXECUTE TASK T1 (TOOL CALL)

1192 **Task Execution:**

1193 *Tool:* wikipedia_search

1194 *Resolved Input Parameters:*

```
1195 1 query: "Kiss and Tell Corliss Archer"
```

1196

1197 *Tool Response (Truncated):*

```
1198 1 Kiss and Tell is a 1945 American comedy film starring Shirley Temple
1199 2 as Corliss Archer. The film was directed by Richard Wallace and
1200 3 produced by Columbia Pictures. In her role as Corliss Archer, Temple
1201 4 portrays a mischievous teenager who becomes involved in a series of
1202 5 comedic misunderstandings. The character of Corliss Archer was
1203 6 originally created by F. Hugh Herbert...
```

```
1204 1 [Additional content about the film plot, production, and reception]
```

1205

1206 **Entity Extractor Invocation:**

1207 *Input:*

```
1208 1 api_response: |
1209 2   Kiss and Tell is a 1945 American comedy film starring Shirley Temple
1210 3   as Corliss Archer. The film was directed by Richard Wallace...
1211 4
1212 5 api_description: |-
1213 6   Search for information about the film "Kiss and Tell" to find the
1214 7   actress who portrayed Corliss Archer
1215 8
1216 9 entities:
1217 10   - name: actress_name
1218 11     description: Name of the actress who portrayed Corliss Archer
1219 12       in Kiss and Tell
1220 13     type: string
```

1221

1222 *Entity Extractor Output:*

```
1223 1 confidence_score: 0.95
1224 2 extracted_entities:
1225 3   actress_name: "Shirley Temple"
1226 4 entities_summary: |-
1227 5   The Wikipedia article explicitly states that "Kiss and Tell is a
1228 6   1945 American comedy film starring Shirley Temple as Corliss Archer."
1229 7   The actress name is clearly identified as Shirley Temple with very
1230 8   high confidence (0.95).
```

1231

1232 **KPI Evaluation:**

- 1233 • Confidence score: $0.95 \geq 0.7$
- 1234 • All required entities extracted: `actress_name` \neq null
- 1235 • **Result:** KPI SUCCESS

1236 **Memory Update:**

```
1237 1 {
1238 2   "T1": {
1239 3     "actress_name": "Shirley Temple"
1240 4   }
1241 5 }
```

1242 **Task Status Update:**

```

1242
1243 1 T1:
2   execution_status: done
3   execution_result:
4   actress_name: "Shirley Temple"
1246

```

C.4 STEP 3: EXECUTE TASK T2 (TOOL CALL) - INITIAL ATTEMPT

Task Execution:

Tool: wikipedia_search

Input Parameter Resolution:

```

1253 Original: <JSON_PATH>T1.actress_name</JSON_PATH>
1254 Memory lookup: T1.actress_name -> "Shirley Temple"
1255 Resolved: "Shirley Temple"

```

Resolved Input Parameters:

```

1258 1 query: "Shirley Temple"
1259

```

Tool Response (Truncated):

```

1261 Shirley Temple Black (April 23, 1928 – February 10, 2014) was an
1262 American actress, singer, dancer, and diplomat. She was Hollywood's
1263 number one box-office draw as a child actress from 1934 to 1938.

```

Early Life and Career:

```

1264 Born in Santa Monica, California, Temple began her film career at
1265 the age of three. She achieved unprecedented popularity in the 1930s,
1266 appearing in films such as Bright Eyes (1934), Curly Top (1935), and
1267 The Little Colonel (1935)...

```

Personal Life:

```

1268 Temple married John Agar in 1945, but they divorced in 1950. She
1269 then married Charles Alden Black in 1950, taking the name Shirley
1270 Temple Black...

```

```

1271 [Response focuses heavily on her acting career and personal life,
1272 with minimal mention of her later diplomatic work]
1273

```

Entity Extractor Invocation:

Input:

```

1274 1 api_response: |
1275 2   Shirley Temple Black (April 23, 1928 – February 10, 2014) was an
1276 3   American actress, singer, dancer, and diplomat...
1277 4   [Full response focused on acting career]
1278
1279 5 api_description: |-_
1280 6   Search for the actress's career information
1281
1282 7 entities:
1283 8   - name: government_position
1284 9     description: Government position held by the actress
1285 10    type: string

```

Entity Extractor Output:

```

1286 1 confidence_score: 0.15
1287 2 extracted_entities:
1288 3   government_position: null
1289 4   entities_summary: |-_
1290 5   The Wikipedia response mentions that Shirley Temple was a "diplomat"
1291 6   but does not specify any particular government position or title.
1292 7   The article focuses primarily on her acting career and personal life.
1293 8   Without a specific government position mentioned, the entity cannot
1294 9   be extracted. Confidence score is very low (0.15) due to lack of
10   specific information.

```

KPI Evaluation:

1296 • Confidence score: 0.15 < 0.7
 1297 • Required entity: government_position = null
 1298 • **Result: KPI FAILURE**

1300 **Memory State (Unchanged):**

```
1302 1   {
1303 2     "T1": {
1304 3       "actress_name": "Shirley Temple"
1305 4     }
1306 5     // T2 failed - no entities stored
1307 6   }
```

1308 **Task Status Update:**

```
1309 1   T2:
1310 2     execution_status: failed
1311 3     execution_result: {}
1312 4     failure_reason: "Entity 'government_position' not found in tool
1313 5       response. Confidence very_low (0.2)."
```

1314 **C.5 STEP 4: RE-PLANNING TRIGGERED**

1315 **Re-planner Agent Invocation:**

1316 *Input:*

```
1318 1   tools:
1319 2     - name: wikipedia_search
1320 3       description: Search Wikipedia for information
1321 4       parameters:
1322 5         - name: query
1323 6           type: string
1324 7           required: true
1325 8
1326 9   query: "What government position was held by the woman who portrayed
1327 10     Corliss Archer in the film Kiss and Tell?"
1328 11
1329 12   current_plan:
1330 13     tasks:
1331 14       - task_id: T1
1332 15         execution_status: done
1333 16         execution_result:
1334 17           actress_name: "Shirley Temple"
1335 18       - task_id: T2
1336 19         execution_status: failed
1337 20         execution_result: {}
1338 21       - task_id: T3
1339 22         execution_status: pending
1340 23
1341 24   failed_task_id: T2
1342 25
1343 26   feedback: |-
```

1344 27 Task T2 failed during entity extraction. The tool response mentioned
 1345 28 that Shirley Temple was a "diplomat" but did not specify any
 1346 29 particular government position or title. The article focused
 1347 30 primarily on her acting career and personal life without detailing
 1348 31 her diplomatic roles. Entity 'government_position' could not be
 1349 32 extracted (null value, confidence very_low = 0.2).

1341 **Re-planner Output (Continuation Plan):**

```
1342 1   tasks:
1343 2     - task_id: T2a
1344 3       task_description: |-
```

1345 4 Search specifically for Shirley Temple's diplomatic career and
 1346 5 government positions with more targeted query

1347 6 task_type: Tool call

1348 7 tool_name: wikipedia_search

1349 8 input_parameters:
1350 9 - name: query
1351 10 type: string
1352 11 value: <JSON_PATH>T1.actress_name</JSON_PATH> diplomat
1353 12 government position Chief of Protocol Ambassador

```

1350      is_reference: true
1351      expected_output_parameters:
1352        - name: government_position
1353          description: |-  

1354            Specific government position title held by Shirley Temple,  

1355            such as Ambassador, Chief of Protocol, or other official  

1356            diplomatic role. Extract the exact position title, not  

1357            just "diplomat".
1358          type: string
1359          dependencies: [T1]
1360          execution_status: pending
1361          execution_result: {}

1362        - task_id: T3a
1363          task_description: |-  

1364            Provide the final answer stating the government position
1365          task_type: Reasoning
1366          tool_name: ""
1367          input_parameters:
1368            - name: position
1369              type: string
1370              value: <JSON_PATH>T2a.government_position</JSON_PATH>
1371              is_reference: true
1372          expected_output_parameters:
1373            - name: final_answer
1374              description: The government position held by the woman
1375              type: string
1376              dependencies: [T2a]
1377              execution_status: pending
1378              execution_result: {}

1379

```

1371
1372 **Plan Merge:** The continuation plan replaces T2 and T3 with T2a and T3a. The system
1373 now proceeds with the updated plan.

1374 **Memory State (Still Unchanged):**

```

1 {
2   "T1": {
3     "actress_name": "Shirley Temple"
4   }
5   // T2 failed - still no new entities
6 }

```

1380
1381 C.6 STEP 5: EXECUTE TASK T2A (TOOL CALL) - RETRY WITH IMPROVED
1382 QUERY
1383

1384 **Task Execution:**

1385 *Tool:* wikipedia_search

1386 *Input Parameter Resolution:*

```

1387 Original: <JSON_PATH>T1.actress_name</JSON_PATH> diplomat government
1388           position Chief of Protocol Ambassador
1389 Memory lookup: T1.actress_name -> "Shirley Temple"
1390 Resolved: "Shirley Temple diplomat government position Chief of
1391           Protocol Ambassador"
1392

```

1393 *Resolved Input Parameters:*

```

1 query: "Shirley Temple diplomat government position Chief of Protocol
1395           Ambassador"
1396

```

1397 *Tool Response (Truncated):*

```

1398 Shirley Temple Black (April 23, 1928 – February 10, 2014) was an
1399 American actress, singer, dancer, and diplomat.
1400
1401 Diplomatic Career:
1402 After her retirement from film and a brief television career, Temple
1403 became actively involved in the Republican Party. Her diplomatic
1404 appointments included:
1405
1406 - United States Ambassador to Ghana (1974-1976): Appointed by

```

```

1404 President Gerald Ford, she served as the U.S. representative to
1405 Ghana during a critical period in African-American relations.
1406
1407 - Chief of Protocol of the United States (1976-1977): In this role,
1408 Temple was responsible for advising the President, Vice President,
1409 and Secretary of State on matters of diplomatic procedure,
1410 organizing state visits, and managing protocol for foreign dignitaries.
1411
1410 - United States Ambassador to Czechoslovakia (1989-1992): Appointed
1411 by President George H. W. Bush, she served during the Velvet
1412 Revolution and the fall of communism in Eastern Europe.
1413
1413 Temple was the first woman to serve as Chief of Protocol and received
1414 numerous awards for her diplomatic service...

```

Entity Extractor Invocation:

Input:

```

1418 1 api_response: |
1419 2   Shirley Temple Black (April 23, 1928 - February 10, 2014)...
1420 3   [Full diplomatic career section as above]
1421 4
1421 5 api_description: |-
1422 6   Search specifically for Shirley Temple's diplomatic career and
1422 7   government positions with more targeted query
1423 8
1423 9 entities:
1424 10   - name: government_position
1425 11     description: |-
1425 12       Specific government position title held by Shirley Temple,
1426 13       such as Ambassador, Chief of Protocol, or other official
1426 14       diplomatic role. Extract the exact position title, not
1427 15       just "diplomat".
1428 16     type: string

```

Entity Extractor Output:

```

1431 1 confidence_score: 0.98
1432 2 extracted_entities:
1433 3   government_position: "Chief of Protocol"
1434 4 entities_summary: |-
1434 5   The Wikipedia response explicitly lists multiple government positions
1435 6   held by Shirley Temple. The article clearly states "Chief of Protocol
1435 7   of the United States (1976-1977)" as one of her positions, along with
1436 8   detailed description of the role. This is extracted with very high
1437 9   confidence (0.98) as it is explicitly stated with the exact title
1437 10  format requested.

```

KPI Evaluation:

- Confidence score: $0.98 \geq 0.7$
- All required entities extracted: $\text{government_position} \neq \text{null}$
- **Result: KPI SUCCESS**

Memory Update:

```

1447 1 {
1448 2   "T1": {
1448 3     "actress_name": "Shirley Temple"
1449 4   },
1450 5   "T2a": {
1450 6     "government_position": "Chief of Protocol"
1451 7   }
1452 8 }

```

Task Status Update:

```

1455 1 T2a:
1456 2   execution_status: done
1456 3   execution_result:
1457 4     government_position: "Chief of Protocol"

```

1458 C.7 STEP 6: EXECUTE TASK T3A (REASONING)
 1459

1460 **Task Execution:**

1461 *Input Parameter Resolution:*

```
1463 Original: <JSON_PATH>T2a.government_position</JSON_PATH>
  Memory lookup: T2a.government_position -> "Chief of Protocol"
  Resolved: "Chief of Protocol"
```

1466 **Reasoning Agent Invocation:**

1467 *Input:*

```
1469 1 task_description: |-
  2   Provide the final answer stating the government position
  3 input_parameters:
  4   - name: position
  5     type: string
  6     value: "Chief of Protocol"
  7 expected_output_parameters:
  8   - name: final_answer
  9     description: The government position held by the woman
 10    type: string
```

1477 *Reasoning Agent Output:*

```
1478 1 Justifications:
  2   1. The original question asks for the government position held by
  3     the woman who portrayed Corliss Archer in "Kiss and Tell"
  4   2. From previous tasks, we identified the actress as Shirley Temple
  5   3. The input parameter provides the government position as
  6     "Chief of Protocol"
  7   4. This is the answer to the original question
  8
 1484 9 execution_result:
 10   status: completed
 11   outputs:
 12     final_answer: "Chief of Protocol"
 13   execution_details:
 14     reasoning_steps:
 15       - "Retrieved government position from input parameters"
 16       - "Verified this answers the original question"
 17       - "Formatted as final answer"
```

1491 **Validation:**

- 1493 • Status: completed
- 1494 • All expected outputs present: final_answer ≠ null
- 1495 • **Result:** Task SUCCESS

1497 **Final Memory Update:**

```
1498 1 {
  2   "T1": {
  3     "actress_name": "Shirley Temple"
  4   },
  5   "T2a": {
  6     "government_position": "Chief of Protocol"
  7   },
  8   "T3a": {
  9     "final_answer": "Chief of Protocol"
 10   }
 11 }
```

1507 **Task Status Update:**

```
1509 1 T3a:
  2   execution_status: done
  3   execution_result:
  4     final_answer: "Chief of Protocol"
```

1512 C.8 EXECUTION SUMMARY
15131514 **Final Answer:** Chief of Protocol
15151516 **Execution Statistics:**
1517

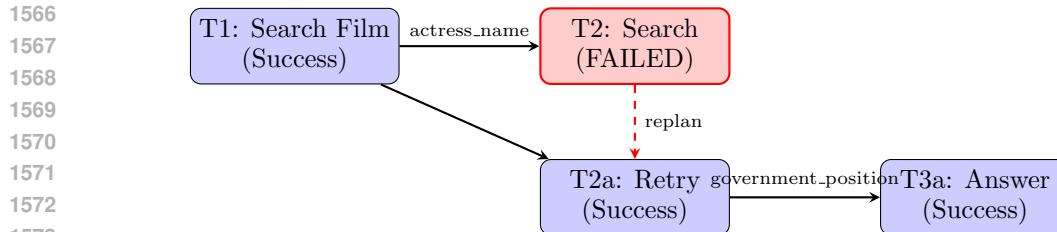
- Total tasks executed: 4 (T1, T2-failed, T2a, T3a)
- Successful tasks: 3 (T1, T2a, T3a)
- Failed tasks: 1 (T2)
- Tool calls: 3 (T1, T2, T2a)
- Reasoning tasks: 1 (T3a)
- Entity extraction invocations: 3 (for T1, T2, T2a)
- Replanning events: 1 (after T2 failure)
- Total LLM calls: 8
 - 1 initial planner
 - 2 tool executions (T1, T2)
 - 2 entity extractions (T1, T2)
 - 1 re-planner
 - 1 tool execution (T2a)
 - 1 entity extraction (T2a)
 - 1 reasoning (T3a)
- Execution status: SUCCESS (after recovery)

1536 **Memory Evolution Timeline:**
15371538 Table 2: Global Memory State Evolution
1539

1540 Step	1541 Event	1542 Memory State
1541 0	1542 Initial	1543 <code>{}</code>
1542 1	1543 T1 success	1544 <code>{"T1": {"actress_name": "Shirley Temple"}}</code>
1543 2	1544 T2 failure	1545 <code>{"T1": {"actress_name": "Shirley Temple"}}</code>
1544 3	1546 Re-planning	1547 <code>{"T1": {"actress_name": "Shirley Temple"}}</code>
1545 4	1548 T2a success	1549 <code>{"T1": {"actress_name": "Shirley Temple"}, "T2a": {"government_position": "Chief of Protocol"}}</code>
1546 5	1551 T3a success	1552 <code>{"T1": {"actress_name": "Shirley Temple"}, "T2a": {"government_position": "Chief of Protocol"}, "T3a": {"final_answer": "Chief of Protocol"}}</code>

1557 **Note:** For clarity and readability, the memory states shown in the timeline above display
1558 only the extracted entities component. In practice, the global memory also maintains the
1559 `current_plan` structure with execution statuses, task descriptions, and results for each
1560 task. This dual structure enables: (1) efficient parameter binding via entity references,
1561 and (2) comprehensive execution context for the re-planner when generating continuation
1562 plans. The re-planner accesses the `current_plan` to understand which tasks succeeded (T1),
1563 which failed (T2), and what entities are available in memory, allowing it to generate T2a
1564 that reuses T1’s output while addressing T2’s specific failure.
1565

1566 **Task Dependency Chain with Re-planning:**



1574 **Key Observations:**

1575

- **Failure Detection:** The entity extractor correctly identified that the initial search for "Shirley Temple" did not provide specific government position information, returning null with very low confidence (0.2).
- **Memory Preservation:** The successfully extracted entity from T1 (`actress_name` = "Shirley Temple") was preserved in global memory throughout the failure and re-planning process.
- **Intelligent Re-planning:** The re-planner analyzed the failure feedback and generated a more targeted search query including keywords like "diplomat", "government position", "Chief of Protocol", and "Ambassador" to improve retrieval.
- **Enhanced Entity Description:** The re-planner improved the entity description in T2a to be more specific: "Extract the exact position title, not just 'diplomat'" which guided the entity extractor to look for precise government position names.
- **Continuation-Based Recovery:** Rather than restarting from scratch, the system continued from the failure point (T2) using T2a, reusing the valid `actress_name` entity from T1's memory.
- **Successful Recovery:** After re-planning with a more specific query, T2a successfully retrieved detailed diplomatic career information and extracted "Chief of Protocol" with very high confidence (1.0).
- **Efficient Parameter Flow:** JSON-path references (`<JSON_PATH>T1.actress_name</JSON_PATH>` and `<JSON_PATH>T2a.government_position</JSON_PATH>`) enabled clean data flow between tasks using the global memory system.
- **Final Answer Accuracy:** The framework successfully recovered from the initial failure and produced the correct answer "Chief of Protocol".

1601 **Comparison to Baseline Approaches:**

1602

- **ReAct:** Would likely continue with vague "diplomat" information without systematic validation, potentially producing an incomplete or incorrect answer.
- **Plan-and-Execute:** Would fail at T2 without structured entity validation and likely restart from scratch rather than preserving T1's successful result, wasting computational resources.
- **KPI-Chain:** Detected the specific failure (missing government position), preserved successful work (actress name), and intelligently adapted the search strategy to succeed on retry.

1612
1613
1614
1615
1616
1617
1618
1619