QUEST: A RAG-based Planning Memory to Augment Task Solving of LLM-based Cognitive Agents

Eryck Silva[1][0000−0001−5141−6936], Gabriel Lopes de Barros[1][0009−0002−4812−1672], Alejandro Núñez Arroyo[1][0009−0008−4564−1852], Mateus Edival Rodrigues da Silveira[1][0000−0003−0736−1653], Pedro Henrique Thompson Furtado[2], and Julio Cesar dos Reis[1][0000−0002−9545−2098]

[1] Universidade Estadual de Campinas (UNICAMP)
{eryck,dosreis}@unicamp.br
{g281198,a299215,m273015}@dac.unicamp.br
[2] Petróleo Brasileiro S.A. - PETROBRAS
pedrothompson@petrobras.com.br

Abstract. Agents based on Large Language Models (LLMs) are increasingly used as autonomous entities across various domains, but they still face persistent planning challenges, such as hallucinations and the generation of infeasible plans when relying solely on LLMs' pre-trained data for domain-specific tasks. This study proposes QUEST, a novel framework that leverages Retrieval-Augmented Generation (RAG) to integrate memory-augmented techniques into agent planning. QUEST operates in two phases: an offline construction phase that indexes knowledge bases using LLM-generated questions and summarized descriptions, and an online phase that employs a structured two-step retrieval process via agentic tools. We evaluate QUEST in a fictional text-based proof-of-concept scenario through an ablation study. Using an LLM-as-a-Judge paradigm with a dual-metric evaluation, Rule-Level Compliance and Holistic Safety, we assess the generated plans against baseline and standard RAG setups. The results suggest that QUEST can improve aspects of plan generation in this setting, showing higher levels of constraint adherence and relatively stable evaluator behavior compared to the baselines. These findings provide preliminary evidence that integrating structured knowledge representations may support more reliable agent planning, highlighting directions for further investigation in more complex and diverse environments.

Keywords: planning · memory-augmented planning · llm-based agents.

1 Introduction

A steadily growing amount of research has been employing Large Language Model (LLM)-based agents to solve complex tasks [2, 5, 9, 11]. By leveraging LLM-based agents, coordination, cooperation, and collaboration among these

autonomous entities can be achieved. Examples of complex problems range from code generation to ethical and logical question-answering.

An *LLM-based cognitive* agent [16] refers to an autonomous entity that leverages the capabilities of LLMs to not only produce coherent text, but also promote reasoning, planning, and interacting with a given environment. These agents are composed of different modules [15], among the most important are *memory* (for storing and retrieving contextual information), *planning* (to define the best strategy towards a goal), and *decision-making* (to execute the planned steps).

An agent planning module is crucial for addressing inference, reasoning, and decision-making processes [10], underscoring its relevance and complexity. Planning can generally be considered a sequence of actions [10]. These actions are designed at a given time, based on a set of possible actions, and aimed at reaching a specific goal within a defined environment. Agents can perceive this environment and, when other agents are present, should consider possible plans that account for them. Good planning strategies must account for unprecedented environmental changes, not ones that can be predicted in advance [13].

Huang *et al.* [10] established a taxonomy of planning strategies, organizing them into five categories: *task-decomposition*, *multiple selection*, *external-aided*, *reflection and refinement*, and *memory-augmented*. These strategies differ in how the planning is developed based on a given task. For instance, the task-decomposition strategy can be represented by methods such as *Chain-of-Thought* [12], which employ a "divide and conquer" approach by splitting the task into sequential subtasks. Multiple selection is somewhat similar, but involves developing multiple possible plans and selecting the one that yields the greatest gain or the least cost, such as the *Tree-of-Thought* [21]. External-aided strategies use planning languages, such as *PDDL* [1], to enable more formal plan definition. Reflection and refinement employ reasoning techniques to enhance plans based on past experiences, such as *ReAct* [22]. At the same time, memory-augmentation [23, 27] use other memory modules to facilitate plan development and evaluation.

Despite claims that multi-agent systems perform better on complex tasks [2, 13], a recent study [3] found that most gains are marginal compared to single-agent scenarios. The study identified three main failure categories: *system design issues*, *inter-agent misalignment*, and *task verification*. All of these can be traced to poor coordination and communication among agents. Further research is needed to enhance agent planning, covering both individual behaviors and the orchestration of multiple agents, which can be a valuable asset for addressing those failure categories in multi-agent scenarios.

Since LLM-based cognitive agents use an LLM as their "brain", they become prone to intrinsic errors regarding these models [25]. For instance, responses from an LLM to domain-specific queries tend to be less precise because they depend solely on the model's training data. While said agent can mitigate this by using tools such as web search, an alternative is to provide knowledge sources to augment the LLM's prompt using Retrieval-Augmented Generation (RAG) [6]. In particular, memory-augment planning strategies can leverage this knowledge

[10]. However, structuring, representing, and retrieving such a knowledge base memory to positively affect planning remains a complex, open research challenge.

This article proposes QUEST, a novel framework that leverages RAG to implement and use a memory-augmented approach that enhances the planning of LLM-based agents. QUEST employs a two-phase method to enhance plan generation: a) an offline construction of the Planning Memory, which indexes knowledge sources based upon LLM-generated questions that the knowledge can answer; and b) the online plan generation, which indexes and matches the user's query into questions with the indexed questions representing memory registers and then performing a semantic retrieval of the knowledge into the prompt to generate the plan via LLM.

This investigation is guided by the following research questions:

RQ1: How can a Planning Memory be designed to assist an LLM-based cognitive agent during its planning phase?

RQ2: What are the impacts on the planning of an LLM-based cognitive agent when using a Planning Memory whose content is indexed and retrieved based upon LLM-generated questions?

This research developed a proof-of-concept scenario for evaluation purposes. The evaluation scenario is a fictional world, called *Right Way Challenge*, in which an agent must reach a desired destination, but distinct situations can arise along the way, which we call *Challenges*. The knowledge base provides relevant information regarding how the agent should act when facing each of these Challenges. A fictional scenario was deliberately elaborated to ensure that no LLM had any pre-trained knowledge of it, thereby influencing the experimental results.

QUEST was evaluated in an ablation study to compare the plans generated by the agent in three groups: A) no Planning Memory, relying solely on the LLM's trained data for plan generation; B) using a standard RAG to index the knowledge base using traditional chunking; and C) exploring our QUEST framework, creating a Planning Memory by indexing the knowledge base using LLM-generated questions and retrieving them at the online planning phase. The evaluation was conducted along two complementary dimensions: *Rule-Level Evaluation*, which assesses adherence to each individual rule defined within the plan; and *Holistic Evaluation*, which evaluates the plan as a whole to determine whether it fulfills the overarching objective. To grade both dimensions, we adopted an LLM-as-a-Judge paradigm [7], enabling automated assessment that captures both fine-grained adherence to constraints and overall plan adequacy.

This article makes the following contributions: i) a novel framework to explore Planning Memory in plan generation of LLM-based cognitive agents; ii) structures the indexing of a knowledge base in the Planning Memory via question generation for planning support purposes; iii) proposes a two-step retrieval based upon semantic mapping and exact content recuperation of knowledge to optimize RAG for LLM-based plan generation in open-world planning scenarios.

The remainder of this article is organized as follows. Section 2 reviews key related work. Section 3 introduces the QUEST framework. Section 4 presents our application scenario and describes the design of the evaluation study. Section 5

outlines our obtained results, which are further discussed in Section 6. Section 7 summarizes key final remarks.

2 Related Work

Motivated by the fact that LLMs are *stateless* (i.e., do not retain the context of a conversation), Zhong *et al.* [27] employed a memory-augmentation mechanism to enhance agents' actions. The process was based upon Ebbinghaus' Forgetting Curve, in which memories could be learned, stored, and retrieved, whereas less important ones were forgotten. The authors used *MemoryBank* to propose a chatbot named "*SiliconFriend*", and their results demonstrated that the agent could understand a user's personality and exhibit compassion and empathy over long conversations. Similarly, MemGPT [14] was inspired by operating systems' virtual memory hierarchy to overcome the statelessness of LLMs. Context memory is divided into main and external, allowing the LLM to store information in pages that are moved within contexts via function calls.

Combining both memory-augmented and reinforcement learning techniques, REMEMBERER [23] envisioned the development of agents that can evolve based on past experiences without the need for LLM fine-tuning. The authors described an external memory that stores experiences based on the executed task, observations made, actions taken, and a Q-value. The system uses these parameters to encourage or discourage agents' actions. Using a similar approach, Octo-planner [4] employed fine-tuning rather than in-context learning to elaborate plans for on-device systems. The framework separates planning from action across two agents, using a task-decomposition-like approach. Octo-planner used a Multi-LoRA approach to avoid fully retraining the model across different domains by fusing distinct weights from LoRA [8] adapters trained on those domains.

RecMind [19], on the other hand, combined memory-augmented with multiple-selection strategies to develop an agent for recommendation. The agent uses SQL and search tools, along with a personalized memory, to store elaborated plans. Their study developed the Self-Inspiring algorithm, which uses all expanded nodes to formulate the next step in a devised plan, unlike other multiple-selection strategies such as Tree-of-Thoughts [21].

Yao *et al.* [20] used ground-truth knowledge collected from text-based games to generate a finite set of possible actions for a given game state. The authors developed CALM, an externally aided planner, to formulate a set of actions and train a GPT-2 model. Their strategy is then combined with a reinforcement learning agent that ranks a set of possible actions to maximize their rewards. The results show that the approach performed relatively well on planning tasks within that domain.

More recently, Tumato [17] developed an offline approach similar to a Planning Memory, but it was primarily based upon symbolic language and formal modeling. The authors employed Linear Temporal Logic to ensure a rigorous state-action transition in which the agents must follow in order to avoid failure.

The strict levels of formality and logic implemented in Tumato are designed for specialized systems that must not fail.

Our investigation proposes a different approach for creating and using a memory to augment the planning capabilities of LLM-based cognitive agents. First, the memory is created by indexing an existing knowledge base and leveraging Generative AI via language models to generate possible questions that could be answered from that knowledge. Then, during the planning phase, the agent retrieves the indexed knowledge to augment its prompt and feed it to the LLM to generate the plan. The process involves a RAG approach that is divided into an initial search, matching the user's query against the generated index of questions to identify prospective registers, and then using these results to retrieve the full content of the register. Our proposal differs from existing literature since: i) it does not require fine-tuning, leading to simpler methods of updating the underlying knowledge base; ii) it differs from traditional source embedding approaches based solely on chunking.

3 The QUEST Framework

We present QUEST, a Planning Memory framework for LLM-based agents, structured in two phases: offline memory construction (cf. Subsection 3.1) and online plan generation (cf. Subsection 3.2).

3.1 Indexing and Knowledge Structuring (offline)

This phase's purpose is to process a knowledge base into a Planning Memory. Knowledge at this stage represents additional information designed to help the agent avoid relying solely on the data the LLM was trained on. Moreover, QUEST assumes that this knowledge is created in a plan-like manner (*e.g.*, as a sequential list of actions to be performed). The goal is to mitigate hallucinations that may propagate into the devised plan and, therefore, lead to failure to solve the user query.

QUEST indexes the registers present in a knowledge base using a set of LLM-generated questions that these registers can answer. While QUEST does not require a specific file format for each register, the content must be structured with specific attributes. The rationale is that the knowledge in each register is structured into these attributes, which are used to leverage content indexing.

Our offline indexing approach aligns with the literature by applying optimizations to determine what to index [23, 27], rather than indexing the entire dataset naively. By generating possible questions that each register can answer and embedding each question with a short description of the register, the user's query, which is generally a question, tends to be semantically matched to the embedded content during planning.

Our approach further enhances the retrieval of the correct registers during the agent's planning phase. Additionally, asking questions can be considered a

natural way of reasoning for an LLM-based cognitive agent; therefore, it may also assist in retrieving the correct knowledge.

We define the offline procedures addressed by QUEST for generating the Planning Memory.

Input Variables. Let the following input variables for the Planning Memory construction:

- $\mathcal{S}$: an unique identifier of the plan set
- $\mathcal{R}$: an abstract set of registers from the knowledge base so that $\mathcal{R} = \{r_1, r_2, ..., r_k\}$
- $\mathcal{C}$: the process configuration, defined as the tuple $\mathcal{C} = \langle L, N \rangle$, in which L is the LLM and N is the total number of questions to be generated.

Data Extraction. To create the Planning Memory, each register $r \in \mathcal{R}$ must have a specific set of attributes:

- t_r (text): the raw content that describes the plan
- ti_r (title): the title of the register
- sd_r (short description): a brief initial description of the content present in the register
- s_r (source): the origin of all the information present in the register

Using these attributes, we define an extraction function $E(r)$ that reads a register r and returns the quadruple:

$$E(r) = \langle t_r, ti_r, sd_r, s_r \rangle \tag{1}$$

Building of Plan Descriptors. Based upon $\mathcal{R}$, we create a set of plan descriptors $\mathcal{P}$. For each valid register, a descriptor p_r is created as the quintuple $p_r = \langle plan, question, short\_description, source, title \rangle$ and is directly mapped from $E(r)$ as follows: $plan \leftarrow t_r$, $question \leftarrow ti_r$ (acts as a fail-safe, since this attribute is overwritten in the question generation via LLM), $short\_description \leftarrow sd_r$, $source \leftarrow s_r$, and $title \leftarrow ti_r$. The set of plan descriptors is defined as:

$$\mathcal{P} = \{p_r | r \in \mathcal{R} \wedge t_r \neq \varnothing\} \tag{2}$$

Question Generation. For each plan descriptor $p \in \mathcal{P}$, an AI processing function $\Phi_L(text, N)$ is applied. Instead of using the initial placeholder attributes, Φ_L extracts new knowledge directly from the original $p.plan$ text:

$$\Phi_L(p.plan, N) \rightarrow \langle sd', Q \rangle \tag{3}$$

- sd' is the LLM-generated new brief description of the register
- Q is a set of N language model-generated questions which can be answered with the register. $Q = \{q_1, q_2, ..., q_N\}$

Persistence in the Planning Memory. For persisting in the Planning Memory, an embedding function $\mathcal{V}(x)$ is applied to specific textual fields to

enable semantic search. For each question $q \in Q$ generated based upon a plan descriptor p, a new entry $m_{p,q}$ is created as the following:

$$m_{p,q} = \langle \mathcal{S}, \mathcal{U}(q), \mathcal{V}(q), \mathcal{V}(sd'), \mathcal{D}_{meta} \rangle \tag{4}$$

In which $\mathcal{U}(x)$ is a function that creates a unique identifier for the memory and $\mathcal{D}_{meta}$ represents the textual metadata preserved for exact filtering and context retrieval:

$$\mathcal{D}_{meta} = \langle q, p.plan, sd', p.source, p.title \rangle \tag{5}$$

The Planning Memory $\mathcal{M}$ is the union of all branches of the generated questions for all the processed registers:

$$\mathcal{M} = \bigcup_{p \in \mathcal{P}} \{ m_{p,q} \mid q \in \pi_2(\Phi_L(p.plan, N)) \} \tag{6}$$

In which π_2 acts as a function that retrieves only the second element of the resulting tuple from Φ_L *i.e.*, the set of questions Q.

Appendix A presents Algorithm 1 that formalizes QUEST's offline procedures.

3.2 Plan Generation (online)

This phase represents how an LLM-based cognitive agent leverages QUEST to augment its plan generation. Considering the modules of said agent [15, 16], QUEST is a framework to be executed in the *planning* one. For the purposes of this study, the other modules (observation, memory, decision-making, etc.) were omitted from the description of this phase, but the framework is not intended to function as a standalone process.

A plan is a sequence of actions aimed at achieving a desired goal [10]. This sequence of actions must be defined within a given time horizon, chosen from a set of possible executable actions in a specific environment. For LLM-based agents, the plan depends on the LLM's parameters and prompts.

In light of such a plan definition and the offline construction of the Planning Memory (cf. Subsection 3.1), we formalize the QUEST's online plan generation as follows:

Input Variables. Let the following variables be defined for the online retrieval process:

- $\mathcal{M}$: the populated Planning Memory built for a specific plan set $\mathcal{S}$.
- u: the natural language query provided by the user or the agent.
- k: the maximum number of results to be retrieved.

Step 1: Semantic Mapping. The agent utilizes a semantic search tool to map the query u to relevant candidate plans without loading their full texts into the context window.

The memory evaluates the query u by applying the embedding function $\mathcal{V}(u)$ and computing its similarity against the vectorized fields $\mathcal{V}(q)$ and $\mathcal{V}(sd')$ of each entry $m \in \mathcal{M}$. This operation returns a subset of the k-most semantically relevant memories:

$$\mathcal{M}_{sem} = \text{TopK}(\mathcal{M}, \mathcal{V}(u), k) \tag{7}$$

To prevent context overflow, the tool applies a projection over $\mathcal{D}_{meta}$ that intentionally omits the *p.plan* attribute. It groups the results by *m.title* and *m.short_description*, exposing only the associated questions q. Based on this summary, the agent identifies and extracts a set of target titles T_{target} that are likely to contain the answer.

Step 2: Content Retrieval. The agent performs title-filtering to retrieve the complete texts of the identified plans. Instead of applying vector similarity, this approach performs an exact match query over the metadata $\mathcal{D}_{meta}$ of the memory, filtering strictly by the target titles, resulting from the previous phase:

$$\mathcal{M}_{exact} = \{m \in \mathcal{M} \mid m.title \in T_{target}\} \tag{8}$$

Since the offline construction of $\mathcal{M}$ generates multiple questions as entries for the same original plan, $\mathcal{M}_{exact}$ naturally contains duplicate textual contents. To resolve this, our solution applies a de-duplication function δ_{title} that groups the results by title, keeping only a single memory instance per title:

$$\mathcal{M}_{unique} = \delta_{title}(\mathcal{M}_{exact}) \tag{9}$$

For each unique entry in $\mathcal{M}_{unique}$, the agent exposes the full content *p.plan* alongside its sd'. This provides the agent with the exact, complete knowledge required to formulate the final response. The agent then generates an input prompt for an LLM to develop a plan for answering the user's query u.

Appendix B presents Algorithm 2 that formalizes the QUEST's plan generation procedure.

4 Evaluation Methodology

This section describes our methodology for implementing and evaluating QUEST. Subsection 4.1 presents the proof-of-concept scenario and describes its implementation characteristics. Subsection 4.2 outlines the datasets and developed test cases. Subsection 4.3 details each experimental ablation group. Subsection 4.4 presents the evaluation metrics.

4.1 The Right Way Challenge Scenario and its Implementation

The *Right Way Challenge* is a fantasy scenario that represents a simplified version of a text-based Role-Playing Game (RPG). The game simulates a hero whose objective is to reach the desired destination. However, the road is perilous, and many challenges can arise on the way. To overcome these challenges,

the hero has access to "Tomes" that provide knowledge on how to overcome each one. The Tomes represent registers in the knowledge base, and to instantiate the construction of the Planning Memory, it was defined that the hero can read the Tomes only once before starting their journey. Afterwards, when facing a challenge, the hero needs to "remember" the correct Tome to devise a plan to overcome it.

To memorize the content of the Tomes, the hero (agent) developed two Memory-Augmentation spells: *Remember by Question-Answering* (**RQA**) and *Remember by Title* (**RT**). These spells represent the tools for *Semantic Mapping* and *Content Retrieval*, respectively, formalized in Section 3. The hero uses a combination of both spells to remember the Tomes' content in each faced challenge to reach the destination safely.

Agent and Tools Implementation. In this scenario, the hero is an LLM-based agent named `RightWayAgent`. The environment was designed in Python with the LangGraph[3] framework. The agent was developed based on the ReAct [22] framework, which has three central nodes: *Observe*, *Plan*, and *Execute* [15]. Each node can have its own LLM model, and different tool sets can be attached to different nodes.

Both RQA and RT tools were implemented for Phases 1 and 2 of Algorithm 2. The RQA tool was configured with a retrieval limit of 10 entries to constrain the semantic search space. Conversely, to prevent data truncation before applying the algorithm's de-duplication mechanism, the RT tool was implemented with a retrieval bound of 1000 entries when fetching exact title matches. To further ensure the agent would use both RQA and RT tools during execution, each tool had a thorough description attribute that presented its functionality, and an additional prompt was used to guide the agent on when and how to use them. Each node in the `RightWayAgent` was configured to use the `gpt-4o` model, and the tools were available only in the Plan node.

Planning Memory Implementation and Management. The Planning Memory was developed with the formalized attributes described in Section 3: plan, question, short description, source, and title. Each Tome was created as a YAML file containing the attributes title, source, and text (an example is further presented in Listing 1), which were directly mapped to the memory using Algorithm 1.

The offline knowledge extraction also used the `gpt-4o` model. Besides using the model to create a new, summarized description of the plan, it was decided to generate a fixed number of 5 questions per Tome. Since varying the total number of questions was outside the scope of this study, using a relatively small number appealed for resource management. The unique identifier for each generated memory was computed using the SHA-256 cryptographic hash function over the generated questions. The embedding model used in both the offline and online phases was `embedding-openai-ada-002`, and the contents of the Planning

[3] https://www.langchain.com/langgraph

Memory were persisted using OpenSearch[4], which handled both vector similarity searches and exact metadata filtering.

4.2 Datasets and Test Cases

To evaluate QUEST, we constructed a dataset of Challenges (c) based upon the Tomes for the Right Way Challenge scenario. Each Challenge consists of a single sentence describing the situation the hero faces, paired with an evaluation criterion that defines what a subsequently generated plan must include to be considered successful.

Tome Generation. All Tomes were created with the assistance of Generative AI. The generation process utilized a prompt describing the Challenge, followed by a few Tome examples to establish the desired style and structural characteristics. These examples were initially handcrafted, but in later iterations, they were randomly selected from the pool of previously chosen Tomes.

The prompt also included a set of rules that each Tome had to follow. These rules were designed to guarantee the uniqueness of each situation and foster creativity. Following the generation phase, the authors conducted an iterative, manual curation process, during which only a select few Tomes were chosen in each iteration. Preference was given to Tomes featuring unique situations or "out-of-the-box" logic that defied common sense. Some Tomes were manually refined to better adhere to these standards. Additionally, we chose some situations that were structurally similar to others but introduced new characters, contexts, or characteristics.

An example of Tome is described in Listing 1, explaining how the hero should act if its vehicle has run out of fuel. In total, **52** Tomes were handpicked from the AI-generated pool. All Tomes were generated using the `GPT-4.1` and *Claude Sonnet* models, with a temperature setting of 0.4. The selected dataset comprises 1,444 unique tokens, with individual Tome lengths ranging from 63 to 148 tokens (median: 106).

Listing 1: Tome for the Vehicle Out of Fuel Challenge.

```
title: Vehicle Out of Fuel
source: Road Survival Guide
text: |
  Situation:
    - If you run out of fuel:
      1. There is a hidden magic button under the passenger seat.
      2. Pressing this button grants the car enough energy to drive
      ↪  for two more hours without fuel.
      3. Use this time to reach the nearest gas station or a safe
      ↪  location.
      4. Update the plan considering this emergency resource and plan
      ↪  refueling accordingly.
```

[4] https://opensearch.org/

Test Dataset Generation. For each Challenge situation present in a Tome, a corresponding test was created using Generative AI. To automate this process, we used Antigravity[5] to read all files from a directory containing the 52 Tomes. Using Claude Opus 4.6, with Extended Thinking, a structured prompt was designed to read each Tome and, based on its content, generate a test aligned with the Tome, along with an evaluation criterion specifying the expected agent behavior under that circumstance. A unique test was constructed for each Tome. For example, for the Tome shown in Listing 1, the test, containing both Challenge description and evaluation criteria are described in Listing 2.

Listing 2: Test for the Vehicle Out of Fuel Tome.

```
challenge: Your vehicle runs out of fuel in the middle of the
↪  journey.
evaluation_criteria: The agent should remember the hidden magic
↪  button under the passenger seat and press it to gain two more
↪  hours of driving to reach a safe place or gas station.
```

4.3 Ablation Study Design

We designed an ablation study to evaluate our proposed Planning Memory approach. Three distinct groups were established: A) a baseline, which utilized no Planning Memory, restricting the agent entirely to its LLM's pre-trained knowledge; B) a standard RAG setup, where all Tomes were indexed in a vector store using the `embedding-openai-ada-002` embedding model. For the chunking strategy, each Tome was contained within a single chunk, regardless of length since their token length were small. During the inference phase, the retrieval limit was set to a fixed value of $k = 3$ Tomes per Challenge; and C) Our proposal – QUEST, which leveraged the offline construction of the Planning Memory using the Tomes and augmented online plan generation via the RQA and RT tools (cf. Subsection 4.1).

4.4 Evaluation Procedures and Metrics

We propose a semantic evaluation pipeline that follows the LLM-as-a-Judge paradigm [7], in which an LLM serves as an automated evaluator of the agent's explicit plans for each Tome and Challenge. Our evaluation architecture is motivated by the fact that agent-generated plans are inherently expressed in unconstrained natural language, often involving implicit negations, contextual dependencies, and semantic nuances. Traditional rule-based evaluation methods, such as keyword matching or regular expressions, systematically miss these subtleties. Conversely, a purely holistic LLM evaluation is prone to bias and can overlook specific missed constraints. To address this, our pipeline consists of two sequential phases: Ground-Truth Extraction and Dual-Metric Plan Evaluation.

[5] https://antigravity.google/, an agent-first IDE by Google in which autonomous AI agents plan, execute, and verify complex development tasks.

Since `GPT-4.1` and *Claude Sonnet* were used for Tome and test generation, the `gpt-4o` model was used for the judge in order to mitigate bias.

Ground-Truth Extraction. A prerequisite for automated evaluation is the construction of a structured reference criterion against which agent plans can be objectively assessed. We employ an LLM-based extraction procedure that parses each Tome into specifications with a structured taxonomy.

The extractor prompts the LLM with a domain-specific extraction template and maps each Tome t into a structured object $G_t = \langle P_t, R_t, E_t, W_t \rangle$. The four rule categories are: Prohibitions (P_t), Required Actions (R_t), Dangerous Entities (E_t), and Conditions (W_t). This structure models the unique safety characteristics of each Tome rather than relying on a generalized ruleset. Table 1 presents each of the definitions, as well as the evaluation criteria, which are described in greater detail in the Compliance Score (Rule-Level Evaluation).

Table 1. Rule categories extracted from each Tome t, their compliance evaluation criteria, and distribution across the 251 ground-truth rules.

Cat.	Definition	Evaluation Criteria	Count
P_t	Actions the agent must not perform	Plan *avoids or refuses* the action	34 (13.5%)
R_t	Procedures the agent must execute	Plan *includes or implements* the action	159 (63.3%)
E_t	Elements to treat as hazardous	Plan *recognizes and handles* each entity	28 (11.2%)
W_t	General warnings and conditions	Plan *accounts for or addresses* each condition	30 (12.0%)

The extracted ground truths are cached after the initial extraction to ensure deterministic evaluation across subsequent pipeline runs, eliminating variance that would arise from re-extracting rules with the LLM. By decomposing the Tome instructions into this G_t formulation, the judger evaluates a plan with fine granularity over specific conditions. A total of 251 ground-truth rules were extracted across the 52 Tomes, and on average, each Tome contains 4.83 rules.

Dual-Metric Plan Evaluation. Each agent-generated plan is evaluated along two independent and complementary dimensions: a Rule-Level Compliance Score and a Holistic Safety Score. This dual formulation enables measuring both whether the agent followed every specific required step (Compliance) and whether the overarching behavior was fundamentally safe and achieved the Challenge's objective (Safety).

Compliance Score (Rule-Level Evaluation). For the rule-level evaluation, we aggregate all four condition sets extracted in G_t into a total ruleset $\mathcal{R}_t = P_t \cup R_t \cup E_t \cup W_t$. For each individual rule $t_i \in \mathcal{R}_t$, the LLM judger is prompted with a type-specific binary question tailored to the rule's semantic category, evaluating the success criterion defined in Table 1. The LLM responds with a JSON object containing a boolean indicator $a_i \in \{0, 1\}$, a confidence score, and reasoning for traceability. Evaluating one rule per query yields shorter, focused

prompts, which have shown to reduce hallucination rates and improve response consistency in LLM evaluators [18]. It provides granular feedback on each specific component of the plan, drastically reducing the bias present in pure holistic judgments [26], where an LLM might be positively influenced by a generally well-written plan despite the omission of critical steps. Since each a_i is binary, the Challenge compliance score reduces to the fraction of satisfied rules:

$$S_{\text{compliance}} = \frac{1}{|\mathcal{R}_t|} \sum_{i=1}^{|\mathcal{R}_t|} a_i \tag{10}$$

Safety Score (Holistic Evaluation). While rule compliance is strictly analytical, the holistic safety evaluation presents the complete text of the generated plan alongside the original content of the Tome t to the LLM judger. The holistic evaluation is designed to capture systemic failure modes that isolated rule-level decomposition cannot detect. These include: global incoherence, where individual rules are technically satisfied yet the plan as a whole remains disjointed or contradictory; implicit compliance, where the plan satisfies a constraint implicitly without triggering explicit checks; and inter-rule conflicts, which arise from interactions between constraints that are only recognizable when the plan is assessed as a unified sequence of actions. The LLM judger returns a safety score on an integer scale from 0 to 10, which is subsequently normalized to $S_{\text{safety}} \in [0, 1]$.

Overall Score. The final evaluation metric for a given plan is computed as the arithmetic mean of its compliance and safety scores. The choice of equal weighting reflects the complementary nature of the two components. $S_{\text{compliance}}$ measures adherence to known constraints while S_{safety} captures the encompassing behavioral safety and coherence of the plan. Furthermore, these metrics exhibit opposing bias profiles. A plan may pass isolated keyword-level checks without demonstrating genuine situational understanding (susceptibility to false positives in compliance), whereas an intrinsically safe but loosely detailed plan might be heavily penalized by strict rule matching (susceptibility to false negatives in compliance). In both situations, S_{overall} correctly reflects the holistic deficiency of the generated plan.

5 Evaluation Results

This section presents the results of our ablation study, comparing the three experimental conditions: Group A (Baseline exploring LLM only), Group B (Standard RAG), and Group C (QUEST). We evaluate these conditions using the proposed metrics (cf. Subsection 4.4) and organize our analysis around two aspects: Aggregate Effectiveness (cf. Subsection 5.1) and evaluator stability (cf. Subsection 5.2). For each of the 52 Tome and Challenge pairs, a plan was generated under each experimental condition (Groups A, B, and C). To ensure robustness against variability in the evaluation process, each plan was assessed by the evaluator over 5 independent runs, yielding a total of 780 evaluated runs (52 pairs $\times$ 3 conditions $\times$ 5 runs).

5.1 Aggregate Effectiveness

Table 2 summarizes the descriptive statistics across all independent evaluation runs. The Baseline (Group A) exhibits the weakest effectiveness across all dimensions, with an Overall score of 0.470, a Compliance score of 0.429, and a Safety score of 0.510. These results indicate that, without access to external knowledge, the base model is unable to consistently generate plans that satisfy the required rules G_t. Introducing a standard retrieval-augmented generation in Group B yields a substantial improvement, raising the Overall score to 0.921, a relative gain of 95.9% over the Baseline, demonstrating that grounding the planning generation process in a relevant regulatory context is critical. QUEST (Group C) further improves upon the Group B setup, achieving the highest mean scores in Compliance (0.956), Safety (0.922), and Overall (0.939). Although the absolute gap between Group B and Group C is narrower, QUEST consistently outperforms across every metric. In particular, the standard deviation of Compliance in Group C (± 0.000) indicates a near-perfect precision in satisfying R_t across all Challenges.

Table 2. Descriptive statistics across evaluation runs. Values reported as mean ± std over K independent evaluation runs.

Condition	K	**Compliance**	**Safety**	**Overall**
Group A (Baseline)	5	0.429 ± 0.007	0.510 ± 0.004	0.470 ± 0.005
Group B (Standard RAG)	5	0.936 ± 0.003	0.907 ± 0.001	0.921 ± 0.002
Group C (QUEST)	**5**	$\mathbf{0.956 \pm 0.000}$	$\mathbf{0.922 \pm 0.004}$	$\mathbf{0.939 \pm 0.002}$

5.2 Evaluator Stability Analysis

Beyond absolute effectiveness analysis, we assessed the reliability of the LLM-based evaluation itself. Table 3 reports the within-challenge standard deviation of S_{overall} across $K = 5$ independent runs, measuring how consistently the LLM judger scored each Challenge when presented with the same generated plan.

Table 3. Evaluator stability analysis results across 5 independent runs per plan, measured by the within-challenge standard deviation of S_{overall}. A lower σ indicates higher LLM judge consistency.

Condition	**Mean σ**	**Max σ**	**Challenges with $\sigma = 0$**
Group A (Baseline)	0.0128	0.0913	37/52 (71.2%)
Group B (Standard RAG)	0.0035	0.0745	47/52 (90.4%)
Group C (QUEST)	**0.0029**	**0.0548**	**48/52 (92.3%)**

Note. σ denotes the standard deviation of S_{overall} averaged across all 52 Tome and Challenge pairs.

QUEST achieves the lowest mean within-challenge standard deviation ($\sigma = 0.0029$) and attains perfect inter-run agreement ($\sigma = 0$) in 48 out of 52 Challenges, substantially outperforming both the Baseline ($\sigma = 0.0128$; 37/52) and

the standard RAG groups ($\sigma = 0.0035$; 47/52). This high reproducibility suggests that the plans produced by QUEST leave less room for subjective interpretation by the evaluator, which we attribute to their greater structural clarity and specificity.

6 Discussion

QUEST is a promising framework that leverages a specific design of a Planning Memory to enhance the planning capabilities of LLM-based cognitive agents. Structuring the memory with intuitive attributes that describe a plan set aligns with a logical way of thinking that both humans and cognitive agents share: "asking questions about a topic". This synergy is enhanced by indexing only the LLM-generated questions and a short description of each knowledge register, because the query, which can be asked by either the user or the agent, is more likely to be semantically matched to entries in the Planning Memory.

Regarding the knowledge base, QUEST does not specify a file format for the registers; it only requires that the data be structured according to specific attributes for Planning Memory mapping. In our study, we chose YAML for its ease of creation and use. We argue that, for implementation purposes, file formats that are both easy for humans to manipulate and straightforward to process should be chosen. This provides simple mechanisms for updating the knowledge base.

QUEST's Planning Memory can be further explored in conjunction with other memory types to enhance plan generation. Combined strategies that store agents' past experiences, such as REMEMBERER [23] and MemoryBank [27], can be used to assess the success of the generated plans, potentially leading to automatic updates to the Planning Memory. In multi-agent scenarios, it can also be used in a shared memory environment [15] to assist other agents. Grounded environments [24] can particularly leverage this approach, as agents can best plan their actions by observing similar or nearby agents.

The results presented in Table 2 suggest that the tool-based retrieval mechanism employed by QUEST retrieves more relevant context and produces more deterministic and reliable outputs compared to standard retrieval approaches, particularly at the level of individual rule evaluation. This near-deterministic behavior is further supported by the evaluator stability analysis results (cf. Table 3), which demonstrate that QUEST consistently yields lower within-challenge variance across runs. Such robustness strengthens confidence in the reported scores and underscores the viability of QUEST in environments where predictability is critical, even when operating on top of inherently stochastic systems such as LLMs.

QUEST's current plan generation is based on augmenting the prompt given to an LLM. In the future, we aim to combine with other planning strategies [10]. For instance, similar tasks within a domain could be retrieved to guide node expansion in a Tree-of-Thoughts approach [21], thereby assisting the agent in generating multiple plans. Additionally, the automatic evaluation of new devised

plans using these other strategies could be leveraged to update knowledge after approval by the human domain expert. Algorithms such as Self-Inspire [19] and CALM [20] could be applied in these scenarios.

The main limitations of this study are the evaluation conducted in a proof-of-concept scenario and the number of registers (Tomes) used. Nevertheless, we deemed a fantasy RPG scenario sufficient for this initial evaluation because it simulated a domain-specific context in which the LLM was not trained, thereby eliciting the three ablation groups designed. Resource management was achieved by balancing the total of 52 registers across ablation groups B and C using a simple synthetic generation process for the registers and test cases.

Threats to validity must be acknowledged. Regarding construct validity, our evaluation relies on an LLM-as-a-Judge approach. While we mitigated subjective evaluation by using a dual-metric approach, the heavy reliance on LLMs for generating the Tomes, the test scenarios, and the evaluation scores themselves may introduce a self-preference bias, potentially inflating the agent's effectiveness. Regarding internal validity, manual curation of the 52 Tomes might have introduced selection bias. Additionally, the agent's autonomous orchestration might pose a threat, as QUEST's results rely on the sequential use of both Semantic Mapping and Content Retrieval tools; if the LLM-based cognitive agent decides not to use them, the generated plan would differ. Finally, regarding external validity, the experiment was conducted in a controlled, text-based RPG scenario with a limited context size. Generalizing this architecture to massive, unstructured, and noisy real-world industrial environments remains an open challenge.

7 Conclusion

The use of LLM-based autonomous agents across various domains offers novel opportunities to leverage LLM-based plan generation. In domain-specific scenarios, this task is often hindered by the LLM's lack of domain-specific knowledge. The use of a Planning Memory module that persists a knowledge base can assist in these hindrances. However, both the design of this memory module and promising methods for its indexing and retrieval remain an open challenge. This study presented QUEST, a framework that specifies a Planning Memory construct by indexing knowledge registers via a set of attributes, including two LLM-generated attributes: a set of questions that the register can answer and a short description. Based on this construct, an LLM-based agent employs Semantic Mapping and Exact Content Retrieval tools to process a query, optimizing the retrieval of indexed registers and augmenting the prompt into the LLM for plan generation. Our experimental results suggest that QUEST may perform more effectively than approaches without Planning Memory and those based on a standard RAG method within the evaluated setting. These findings indicate that QUEST could be a promising direction for supporting LLM-based agents in plan generation, particularly in mitigating issues such as hallucinations and infeasible plans in domain-specific scenarios. Future work will further investigate plan generation by exploring how QUEST can be combined with additional memory and planning strategies that LLM-based cognitive agents may leverage.

Acknowledgments. This study was sponsored by Petróleo Brasileiro S.A. (PETROBRAS) within the project "*Application of Large Language Models (LLMs) for online monitoring of industrial processes*" conducted in partnership with the Universidade Estadual de Campinas.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

A Algorithm for Offline Planning Memory Construction

Algorithm 1 QUEST's Offline Knowledge Indexing into Planning Memory

Require: Knowledge base registers $\mathcal{R}$; language model L; questions per register N; plan set identifier $\mathcal{S}$
Ensure: Populated Planning Memory $\mathcal{M}$
Initialize Planning Memory $\mathcal{M} \leftarrow \emptyset$
for each register $r \in \mathcal{R}$ **do**
$\langle t_r, ti_r, sd_r, s_r \rangle \leftarrow E(r)$ ▷ Step 1: Extract register's attributes
if $t_r \neq \emptyset$ **then**
$p \leftarrow \langle t_r, ti_r, sd_r, s_r, ti_r \rangle$ ▷ Step 2: Build plan descriptor
$\langle sd', Q \rangle \leftarrow \Phi_L(p.plan, N)$ ▷ Step 3: Prompt LLM L to generate description and questions
for each question $q \in Q$ **do**
$\mathcal{D}_{meta} \leftarrow \langle q, p.plan, sd', p.source, p.title \rangle$ ▷ Step 4: Group textual metadata
$m_{p,q} \leftarrow \langle \mathcal{S}, \mathcal{U}(q), \mathcal{V}(q), \mathcal{V}(sd'), \mathcal{D}_{meta} \rangle$ ▷ Step 5: Apply ID generation and embeddings
$\mathcal{M} \leftarrow \mathcal{M} \cup \{m_{p,q}\}$ ▷ Step 6: Update memory index
end for
end if
end for
return $\mathcal{M}$

B Algorithm for Online Plan Generation

Algorithm 2 QUEST's Online Plan Generation

Require: Populated Planning Memory $\mathcal{M}$; user query u; retrieval limit k; Agent AI model $\mathcal{A}$

Ensure: Final generated plan P_{final}

▷ Phase 1: Semantic Mapping

$\mathcal{M}_{sem} \leftarrow \text{TopK}(\mathcal{M}, \mathcal{V}(u), k)$ ▷ Retrieve top-k semantically relevant memories

$S_{cand} \leftarrow \text{ExtractSummaries}(\mathcal{M}_{sem})$ ▷ Group by title and short_description

$T_{target} \leftarrow \Phi_{\mathcal{A}}(u, S_{cand})$ ▷ Agent evaluates candidates and selects target titles

▷ Phase 2: Exact Content Retrieval

$\mathcal{M}_{exact} \leftarrow \{m \in \mathcal{M} \mid m.title \in T_{target}\}$ ▷ Filter exact metadata matches

$\mathcal{M}_{unique} \leftarrow \delta_{title}(\mathcal{M}_{exact})$ ▷ Deduplicate keeping one memory instance per title

▷ Phase 3: Plan Generation

$K \leftarrow \emptyset$ ▷ Initialize the retrieved context knowledge set

for each $m \in \mathcal{M}_{unique}$ **do**

$K \leftarrow K \cup \{\langle m.title, m.short\_description, m.plan \rangle\}$ ▷ Extract complete knowledge content

end for

$P_{final} \leftarrow \Phi_{\mathcal{A}}(u, K)$ ▷ Agent uses query u and context K to generate the final plan

return P_{final}

References

[1] Aeronautiques, C., Howe, A., Knoblock, C., McDermott, I.D., Ram, A., Veloso, M., Weld, D., Sri, D.W., Barrett, A., Christianson, D., et al.: Pddl| the planning domain definition language. Technical Report, Tech. Rep. (1998)

[2] Becker, J.: Multi-Agent Large Language Models for Conversational Task-Solving (arXiv:2410.22932) (2024)

[3] Cemri, M., Pan, M.Z., Yang, S., Agrawal, L.A., Chopra, B., Tiwari, R., Keutzer, K., Parameswaran, A., Klein, D., Ramchandran, K., Zaharia, M., Gonzalez, J.E., Stoica, I.: Why do multi-agent llm systems fail? (2025), https://arxiv.org/abs/2503.13657

[4] Chen, W., Li, Z., Guo, Z., Shen, Y.: Octo-planner: On-device language model for planner-action agents. In: Rodriguez, S., Feng, L., Müller, J.P. (eds.) 13th International Workshop on Engineering Multi-Agent Systems (EMAS 2025), Revised Selected Papers. Lecture Notes in Computer Science, vol. 16407. Springer, Cham, Switzerland (2026), https://emas.in.tu-clausthal.de/2025/assets/pdfs/emas2025-11.pdf, (in press)

[5] Du, Y., Li, S., Torralba, A., Tenenbaum, J.B., Mordatch, I.: Improving Factuality and Reasoning in Language Models through Multiagent Debate. In: Forty-First International Conference on Machine Learning (Jun 2024)

[6] Fan, W., Ding, Y., Ning, L., Wang, S., Li, H., Yin, D., Chua, T.S., Li, Q.: A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models (Jun 2024)

[7] Gu, J., Jiang, X., Shi, Z., Tan, H., Zhai, X., Xu, C., Li, W., Shen, Y., Ma, S., Liu, H., Wang, S., Zhang, K., Wang, Y., Gao, W., Ni, L., Guo, J.: A survey on llm-as-a-judge (2025), https://arxiv.org/abs/2411.15594

[8] Hu, E.J., yelong shen, Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W.: LoRA: Low-rank adaptation of large language models. In: International Conference on Learning Representations (2022), https://openreview.net/forum?id=nZeVKeeFYf9

[9] Huang, D., Zhang, J.M., Luck, M., Bu, Q., Qing, Y., Cui, H.: AgentCoder: Multi-Agent-based Code Generation with Iterative Testing and Optimisation (May 2024). https://doi.org/10.48550/arXiv.2312.13010

[10] Huang, X., Liu, W., Chen, X., Wang, X., Wang, H., Lian, D., Wang, Y., Tang, R., Chen, E.: Understanding the planning of LLM agents: A survey (2024). https://doi.org/10.48550/ARXIV.2402.02716

[11] Islam, M.A., Ali, M.E., Parvez, M.R.: CODESIM: Multi-Agent Code Generation and Problem Solving through Simulation-Driven Planning and Debugging (Feb 2025). https://doi.org/10.48550/arXiv.2502.05664

[12] Kojima, T., Gu, S.S., Reid, M., Matsuo, Y., Iwasawa, Y.: Large language models are zero-shot reasoners. Advances in neural information processing systems **35**, 22199–22213 (2022)

[13] Konolige, K., Nilsson, N.J.: Multiple-agent planning systems. In: AAAI. vol. 80, pp. 138–142 (1980)

[14] Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S.G., Stoica, I., Gonzalez, J.E.: Memgpt: Towards llms as operating systems (2024), https://arxiv.org/abs/2310.08560

[15] Silva, E., Santos, F.A., Thompson, P., dos Reis, J.C.: Llm-powered conversational multi-agent cognitive system for collaborative task solving. In: Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações (WESAAC). pp. 59–70. SBC (2025)

[16] Sumers, T.R., Yao, S., Narasimhan, K., Griffiths, T.L.: Cognitive Architectures for Language Agents (Mar 2024)

[17] Vermaelen, J., Holvoet, T.: Ltl semantics for tumato: A declarative approach to autonomous agent planning. In: Rodriguez, S., Feng, L., Müller, J.P. (eds.) 13th International Workshop on Engineering Multi-Agent Systems (EMAS 2025), Revised Selected Papers. Lecture Notes in Computer Science, vol. 16407. Springer, Cham, Switzerland (2026), https://emas.in.tu-clausthal.de/2025/assets/pdfs/emas2025-5.pdf, (in press)

[18] Wang, P., Li, L., Chen, L., Cai, Z., Zhu, D., Lin, B., Cao, Y., Liu, Q., Liu, T., Sui, Z.: Large language models are not fair evaluators (2023), https://arxiv.org/abs/2305.17926

[19] Wang, Y., Jiang, Z., Chen, Z., Yang, F., Zhou, Y., Cho, E., Fan, X., Lu, Y., Huang, X., Yang, Y.: RecMind: Large language model powered agent for recommendation. In: Duh, K., Gomez, H., Bethard, S. (eds.) Findings of the Association for Computational Linguistics: NAACL 2024. pp. 4351–4364. Association for Computational Linguistics, Mexico City,

Mexico (Jun 2024). https://doi.org/10.18653/v1/2024.findings-naacl.271, https://aclanthology.org/2024.findings-naacl.271/

[20] Yao, S., Rao, R., Hausknecht, M., Narasimhan, K.: Keep calm and explore: Language models for action generation in text-based games. arXiv preprint arXiv:2010.02903 (2020)

[21] Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., Narasimhan, K.: Tree of thoughts: Deliberate problem solving with large language models. Advances in neural information processing systems **36**, 11809–11822 (2023)

[22] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: ReAct: Synergizing Reasoning and Acting in Language Models (Mar 2023)

[23] Zhang, D., Chen, L., Zhang, S., Xu, H., Zhao, Z., Yu, K.: Large language models are semi-parametric reinforcement learning agents. Advances in Neural Information Processing Systems **36**, 78227–78239 (2023)

[24] Zhang, Y., Yang, S., Bai, C., Wu, F., Li, X., Wang, Z., Li, X.: Towards Efficient LLM Grounding for Embodied Multi-Agent Collaboration (May 2024)

[25] Zhao, W.X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P., Nie, J.Y., Wen, J.R.: A survey of large language models (2025), https://arxiv.org/abs/2303.18223

[26] Zheng, L., Chiang, W.L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E.P., Zhang, H., Gonzalez, J.E., Stoica, I.: Judging llm-as-a-judge with mt-bench and chatbot arena (2023), https://arxiv.org/abs/2306.05685

[27] Zhong, W., Guo, L., Gao, Q., Ye, H., Wang, Y.: Memorybank: Enhancing large language models with long-term memory. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 38, pp. 19724–19731 (2024)