

Reinforcement Learning Problem Solving with Large Language Models

Sina Gholamian

Thomson Reuters AI Labs

Toronto, Canada

sina.gholamian@thomsonreuters.com

Domingo Huh

Thomson Reuters AI Labs

Toronto, Canada

domingo.huh@thomsonreuters.com

Abstract

Large Language Models (LLMs) encapsulate an extensive amount of world knowledge, and this has enabled their application in various domains to improve the performance of a variety of Natural Language Processing (NLP) tasks. This has also facilitated a more accessible paradigm of conversation-based interactions between humans and AI systems to solve intended problems. However, one interesting avenue that shows untapped potential is the use of LLMs as Reinforcement Learning (RL) agents to enable conversational RL problem solving. Therefore, in this study, we explore the concept of formulating Markov Decision Process-based RL problems as LLM prompting tasks. We demonstrate how LLMs can be iteratively prompted to learn and optimize policies for specific RL tasks. In addition, we leverage the introduced prompting technique for episode simulation and Q-Learning, facilitated by LLMs. We then show the practicality of our approach through two detailed case studies for “Research Scientist” and “Legal Matter Intake” workflows.

1 Introduction

The recent inception of Generative Large Language Models into the limelight has significantly influenced and reshaped the applications of Natural Language Processing (NLP) and the ways that LLMs contribute to it (Zhao et al., 2023). Because LLMs are trained on a massive set of internet data, this comprehensive knowledge gives them the capabilities to extract semantics and respond coherently in a human-style conversation. Furthermore, fine-tuning through Reinforcement Learning with Human Feedback (RLHF) (Ouyang et al., 2022) has enabled LLMs to accurately follow input prompt instructions. These capabilities have allowed LLMs to be used for a variety of out-of-the-box applications, including chatbot agents (Dan et al., 2023; Limna et al., 2023), coding (Vaithilingam et al.,

2022; OpenAI, 2023), and general common-sense reasoning (Wei et al., 2022). In addition, some studies have shown that LLMs can be further leveraged for knowledge reasoning in professional domains such as in medical (Liévin et al., 2022) and legal (Hakimi Parizi et al., 2023) fields.

In RL problems, one goal can be to optimize an objective function (e.g., the sum of the expected long-term rewards) as an agent interacts with the environment and receives feedback. Deep RL (Mnih et al., 2015), a paradigm that implements the agent or policy with a deep learning model, has gained attention as a function approximator to implement RL agents and policies. Once the Deep RL model is trained, it can receive a set of observations as inputs, and output the corresponding action to navigate the agent to the next state. RL agents, and their policy implementations as RL models, do not necessarily have well-defined natural language interfaces, which limits their implementation accessibility to non-technical users. To this aim, prior work (Woodward et al., 2020) has explored the advantages of expanding the RL agent interfaces to unstructured formats and has achieved improved task performance.

In parallel, LLM-based prompting has introduced a new interaction paradigm between AI systems and end-users that is more intuitive and accessible to every-day and non-technical users. In addition, recent developments in prompting strategies, such as Chain-of-Thought (CoT) prompting (Wei et al., 2022), have improved LLMs’ abilities to decompose complex tasks into manageable sub-steps and further enhance their reasoning and planning capabilities. This motivates the exploration of whether RL problems can be interactively solved through iterative prompting of the LLMs. Based on a set of input and output requirements for the RL task at hand, the LLM can arguably execute the task, evaluate whether the requirements are fully met, and determine if additional iterations

are required to satisfy all the requirements. Subsequently, the interesting question of whether the embedded knowledge and reasoning capabilities of LLMs can be leveraged through iterative prompting to allow LLMs to function as RL agents remains unanswered.

Consequently, in this research, we propose a new framework that leverages the reasoning and problem-solving capabilities of LLMs to align them for RL problem-solving. For this process, we leverage the latest paradigm of human and AI systems interactions, i.e., natural language prompting of LLMs. More specifically, we make the following contributions:

- We introduce an iterative prompting strategy for communicating with LLMs and formulate RL problems based on Markov Decision Process and Q-Learning into LLM prompting tasks. We then iteratively leverage the LLM’s knowledge and self-reflection to optimize the RL problem and extract the desired output from the LLM.
- We propose the integration of episode generation and simulation into the prompting chain and thus enable LLM-based policy learning, and, subsequently, we solicit the optimal policy outcomes (i.e., episodes) from the LLM.
- We provide two detailed case studies of how our approach can be effectively implemented to extract optimal policies for the Research Scientist and Legal Matter Intake workflows.

2 Prior Art

While the primary applications of LLMs are in NLP tasks, and are receiving significant attention from the research community, the applications of LLMs in adjacent areas such as Reinforcement Learning (RL) have the potential to grow (Wang et al., 2023; Xi et al., 2023). Previously, Janner et al. (Janner et al., 2021) proposed an RL formulation via a sequence-to-sequence learning model based on the transformers architecture. However, this modeling lacks the more accessible natural language and prompt-based interactions with LLMs and human users. Consequently, recent studies have explored building RL agents with LLMs or enabling interactions between LLMs and RL agents to improve the accomplishment of specific tasks, to name a few, robot control (Hu et al., 2023) and game playing (Xu et al., 2023).

Prior work has shown prompt engineering scenarios can help in decomposing complex tasks and thus reach better outcomes with LLMs. Wei et al. (Wei et al., 2022) introduced Chain-of-Thought (CoT) prompting, which prompts the LLM to solve a task in a step-by-step manner. Similarly, Tree-of-Thoughts (ToT) (Yao et al., 2023) and Graph-of-Thoughts (GoT) (Besta et al., 2023) expand on CoT and create tree and graph structures of prompts, such that different paths of the tree can break down the task into varied sets of smaller substeps. On an adjacent track, self-reflection prompting (Shinn et al., 2023) aims to iteratively refine the output of the LLM and get closer to the desired output. Our approach is different from the aforementioned prompting methods as we introduce a framework for solving RL tasks with iterative prompting techniques that take into consideration the requirements of reaching an optimal RL policy.

In essence, compared to prior approaches, we argue that our approach to successfully implementing natural language and prompt-based interactions with LLMs would be a more intuitive and potentially transformative method for interactions between human users and AI systems. Thus, it naturally represents the forthcoming paradigm for optimizing RL problems.

3 Preliminaries

An RL problem is often formally defined as a Markov Decision Process (MDP) (Puterman, 2014), where MDP provides a mathematical definition of the environment that the RL agent is interacting with. An MDP is defined as a tuple of (S, A, P, R, γ) , where $S = \{s_1, s_2, \dots, s_n\}$ denotes a set of states that an agent can be within the environment; $A = \{a_1, a_2, \dots, a_m\}$ represents a set of actions that agent can take at each state; P is a state transition probability matrix that given the current state s and action a at time t , represents the probability of ending up in state s' at the next time step, i.e., $P(s'|s, a) = \Pr(S_{t+1} = s' \mid S_t = s, A_t = a)$; R is a reward function that represents the expected reward for taking action a in state s and moving to state s' , $R(s, a, s') = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s']$; $\gamma \in [0, 1]$ is a discount factor that encourages more immediate rewards for $\gamma < 1$. Based on this framework we can define a policy (π) that is a strategy (i.e., function) that the agent can leverage to decide the next action to take based on the current state:

$$\pi(a|s) = Pr(A_t = a \mid S_t = s).$$

3.1 Q-Learning

One popular RL algorithm is Q-Learning which uses a Q-table to store the expected rewards for state-action pairs. The agent then uses this table to select the best action based on its current state. The Q-value update formula that chooses the action that maximizes the expected sum of rewards at time step t is defined as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \times [r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

where α is the learning rate, γ is the discount factor, r_{t+1} is the reward received after taking action a_t at state s_t , and $\max_a Q(s_{t+1}, a)$ represents the expected reward for the action that results in maximum reward at the next state. This algorithm can be augmented to balance both exploitation and exploration through ϵ -greedy approach. With probability $p = \epsilon$, the agent takes a randomly chosen action from the set of all possible actions A to encourage exploration, and otherwise selects the action that maximizes the reward, i.e., exploitation:

$$a_t = \begin{cases} a \sim \text{Uniform}(A) & : p = \epsilon \\ \arg \max_a Q(s_t, a) & : p = 1 - \epsilon \end{cases}$$

4 Methodology

We propose utilizing LLMs to formulate the RL problem as a prompting task and then leverage the internal knowledge of LLMs to train a Q-Learning RL agent, and thereby optimize the desired policy. Initially, it is necessary to clearly translate and define the communication of the requirements for an RL problem to an LLM in text-based prompts. Therefore, we define accurate steps to transform an RL problem into a set of prompts to elicit the desired outputs, in this case, the optimized policy, from the LLM.

4.1 Prompt Framework

Based on the preliminaries discussed in the previous section for defining an RL problem within the MDP framework, we first outline a framework to provide the RL problem requirements through different sections of the prompt. Specifically, based on the tuple (S, A, P, R, γ) , we initially establish the context of the RL problem, followed by detailing states, actions, and rewards. We then compile these necessary steps in two elaborative case studies.

Context Setup. We define the problem context for the LLM, which sets the expected behavior of the LLM and outlines the inputs for the MDP-based RL problem. This context setup is crucial as it aligns the LLM to interpret and respond to the RL problem requirements.

Problem Context: *You are an RL agent tasked with maximizing cumulative reward for a given task. You will be provided with the task, states, possible actions at each state, and rewards.*

Task. We then specify the RL task that LLM is intended to optimize. The placeholder provided below indicated that it can be filled in with any arbitrary task. For example, the task targeted for optimization could be the “*Workflow of a research scientist*”.

Task: *{place_holder}*

Inputs. Once the task is specified, we include the necessary inputs to formulate the task as a MDP-based RL problem. We specifically provide details on states, actions, and rewards. It is important to note that additional inputs such as γ and ϵ can also be provided if required. If these parameters are not specified, the LLM will decide them.

States: $\{s_1, s_2, \dots, s_n\}$
Possible actions: $\{a_1, a_2, \dots, a_m\}$
Rewards: $\{r_1, r_2, \dots, r_k\}$

Requirements. This section can be added if specific task-related requirements are necessary. As a part of the requirements, we include episode simulation with the LLM. RL agents require episodes of interactions with the environment to receive rewards and thereby evaluate different decisions. However, real-world interactions can often be time-consuming and costly. Hence, in line with our concept of utilizing LLMs as RL optimizers and agents, we also leverage LLMs as environment simulators. More specifically, in this setup, we request the LLM to generate episodes that begin from the start state and conclude at a terminal state. These episodes are then utilized for Q-Learning with the LLM. Below, we have listed some example requirements for Q-Learning and we provide more detailed case studies in the later sections.

Requirements:

1. Solve it with Q-Learning.
2. Simulate the environment for $\{n\}$ episodes.
3. Episodes MUST begin at "Start" and finish at "End".
4. ...

Outputs. We then instruct the LLM regarding the expected output. We ask the LLM to return the Q table and values for the optimal episode. This way we can evaluate the effectiveness of our framework in achieving the optimal policy.

Output: *Print the Q-table, and list the state/action pairs and their "Q-values" for the optimal episode from "Start" to "End".*

Iterative Check. This step revisits the output of the LLM from the last iteration to confirm if it meets the requirements. Iterations can continue until the desired output or the maximum number of trials is reached. Similar approaches, like self-reflection (Shinn et al., 2023), have been shown to improve the alignment with the intended task and elicitation of desired outputs from LLMs.

Did your output satisfy all of the following requirements? If not, you MUST take a fresh approach and execute it if necessary.

{repeat "Task/States/Actions/Rewards" and "Requirements" from above}

4.2 Algorithm

Algorithm 1 outlines the pseudocode for algorithmically implementing RL problem formulation to solve an arbitrary RL optimization problem. First, task-specific inputs are defined, followed by the crafting of a generic prompt incorporating these inputs, *GeneratePrompt(.)* at Line 4. We then iterate in the while loop on Lines 6-11 to check whether the requirements are met or the maximum number of iterations is reached. If the LLM-generated output does not meet the requirements, the LLM is re-prompted with the Requirements U on Line 7. Finally, the output from the LLM, which should be the Q-table and the optimal episode for the learned policy according to the RL task, is returned. This algorithm can be generally applied to any prompt-based LLM to iteratively find a solution to RL problems.

Algo. 1: RL problem solving with LLMs

Input: Problem Context C , Task T , States S , Actions A , Rewards R , Requirements U , and iterations M

Output: Q-Table for Policy π

```

1 Satisfied  $\leftarrow$  False;
2 Max_Iter  $\leftarrow$   $M$ ;
3 iter  $\leftarrow$  0;
4 Prompt  $\leftarrow$ 
   GeneratePrompt( $C, T, S, A, R, U$ );
   /* e.g., Figure 4 */
5 Output  $\leftarrow$  LLM(Prompt);
6 Prompt  $\leftarrow$ 
   GeneratePrompt( $T, S, A, R, U$ );
   /* e.g., Figure 5 */
7 while iter < Max_Iter do
8   Satisfied  $\leftarrow$ 
     LLM(Prompt, Output);
9   if Satisfied then
10    | return Output;
11  else
12    | Output  $\leftarrow$  LLM(Prompt);
13    | iter  $\leftarrow$  iter + 1
14 return Output;

```

5 Case Studies

In this section, we provide two illustrative case studies on how our approach can be applied to optimize different workflows.

5.1 Research Scientist Workflow

In this case study, we consider our task to be optimizing the workflow of a Research Scientist. The professional is involved in daily research activities like literature reviews and research publications. The goal is to learn and optimize the sequence of steps that a researcher should take to optimize their workflow.

To address this problem with Q-Learning through LLM agents, we first create a simulation of the environment based on the provided input states, actions, and rewards. Then, the Q-Learning algorithm is implemented and iterated for a predefined number of episodes (e.g., 1000) to optimize the policy. Each episode begins from the initial state, i.e., 'Start', and ends at the terminal state, i.e., 'End'. To ensure convergence, other parameters, such as the maximum number of steps per episode can also be defined. After training, the learned policy is stored in the Q-table and we can identify and extract episodes with the highest cumulative

rewards. As an example, we provide the following states, actions, and rewards to be included in the LLM’s prompt. For simplicity, we assume equal probabilities for existing transitions from one state to another. Later, we will discuss how this condition can be relaxed by deriving the probabilities from actual workflows.

States = { ‘Start (ST)’, ‘Initiate Research (IR)’, ‘Literature Review (LR)’, ‘Experiment Plan (EP)’, ‘Experiment Execution (EE)’, ‘Data Analysis (DA)’, ‘Manuscript Drafting (MD)’, ‘Submission to Venue (SV)’, ‘Revision of Manuscript (RM)’, ‘Peer Review (PR)’, ‘Result Publication (RP)’, ‘End (ED)’ }

Actions = { ‘ST’: [‘IR’], ‘IR’: [‘LR’, ‘EP’], ‘LR’: [‘EP’, ‘MD’], ‘EP’: [‘EE’], ‘EE’: [‘DA’], ‘DA’: [‘MD’, ‘EP’], ‘MD’: [‘SV’, ‘RM’], ‘SV’: [‘RM’, ‘PR’], ‘RM’: [‘EE’, ‘SV’], ‘PR’: [‘RM’, ‘RP’], ‘RP’: [‘ED’], ‘ED’: [‘ED’], ‘ELSE’: -inf }

Additionally, we assign rewards accordingly to optimize the workflow. In this scenario, we aim to reach the ‘End’, i.e., the terminal state, as soon as possible. Hence, for simplicity, we assign a reward of -1 to every state, except the terminal state, which receives a reward of 0 .

$$r(s) = \begin{cases} -1 & \text{if } s \neq \text{‘End (ED)’} \\ 0 & \text{if } s = \text{‘End (ED)’} \end{cases}$$

Rewards = { ‘ED’: 0 , else: -1 }

Lastly, we define the requirements for optimizing the task of “Research Scientist Workflow” as follows.

Requirements:

1. Solve it with Q-Learning.
2. First, simulate the environment for 1000 episodes and fill in the Q-table.
3. Episodes MUST begin at "Start" and finish at "End".
4. ...

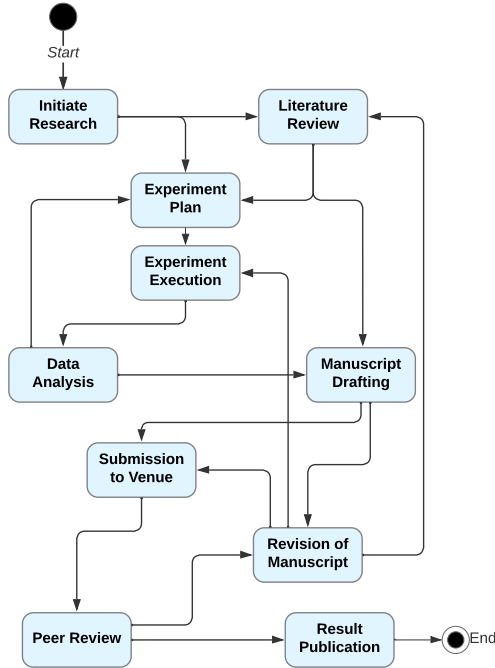


Figure 1: Research Scientist workflow.

Similarly, we define the possible actions for each state as follows. For every state s we list all possible actions a that can be taken from that state, $\forall s \in S: [a_{p1}, \dots, a_{pn}]$, and for simplicity, we assume that actions from each state initially have the same probability. States and action combined define the state machine for the MDP. Figure 1 depicts the state machine for the research scientist workflow. While we have aimed to represent a meaning flow in this case study, it is for demonstration purposes and may not necessarily fully generalize to real-life scenarios.

5.2 Legal Matter Intake Workflow

As the second case study, we illustrate the workflow for tracking legal matters. This flow outlines the steps that legal associates within law firms typically follow when receiving a new legal request from a client and managing it through to the completion of the matter. The flow generally consists of the following steps:

1. Matter Intake (MI): The first step involves taking in the client’s request and assessing whether the firm has the required expertise and resources to undertake this matter. **2. Conflict Assessment**

(CA): Upon intake, the firm or attorney involved is required to assess conflicts of interest to ensure they are not precluded from representing the potential client. In some cases, CA can be bypassed, for example, for existing clients. **3. Initial Assessment (IA):** An associate within the legal firm then conducts an initial assessment to identify client’s case strengths and weaknesses. The associate will also come up with a plan on how to proceed with the case. This step may circle back to CA if further information are required after the initial assessment. **4. Client Communication (CC):** In this step, the legal firm communicates with the potential client, briefing them on the outcomes of the initial assessment and providing an overview of the next legal steps. This also give an opportunity to the client to discuss any questions they may have. **5. Fee and Payment (FP):** This step involves communicating the legal fees or estimations associated with the case to the client, such as hourly rates or any contingency fees. This step can be potentially bypassed assuming the client is already aware of the law firm’s fees. **6. Proposal Preparation (PP):** Based on the initial assessment and the client’s needs, the legal professional prepares a proposal that outlines the scope of work, the estimated cost, and the expected outcome. **7. Proposal review (PR):** Once the proposal is prepared, the legal professional reviews it with the client and address any questions or concerns. After the client agrees with the plan, the matter progresses to the Case Management step. **8. Case Management (CM):** This step contains the bulk of the legal research work. CM entails that the legal professional will conduct legal research, draft legal documents, and potentially represent their client in court or other legal proceedings. **9. Billing (BI):** At last, the law firm manages the billing and invoicing process, which includes tracking the time spent on the case, i.e., billable hours, preparing invoices, and sending them to the client for payment.

Based on the above-mentioned possible stages of the legal matter intake process, we produce a feasible arrangement of states and potential transitions among them in Figure 2. It is important to reiterate that this workflow is intended for demonstration purposes only and might not accurately capture the potential variability of this process. Similar to the research scientist workflow, and based on Figure 2, **States** and **Actions** for legal matter intake workflow are outlined below.

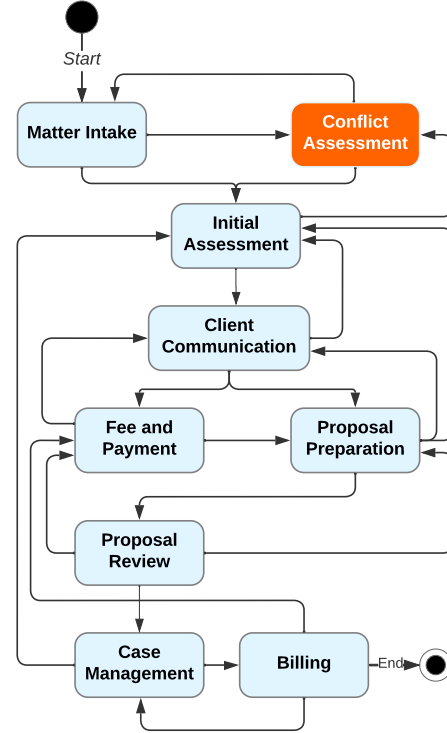


Figure 2: Legal Matter Intake workflow.

States = { ‘Start (ST)’, ‘Matter Intake (MI)’, ‘Conflict Assessment (CA)’, ‘Initial Assessment (IA)’, ‘Client Communication (CC)’, ‘Fee and Payment (FP)’, ‘Proposal Preparation (PP)’, ‘Proposal Review (PR)’, ‘Case Management (CM)’, ‘Billing (BI)’, ‘End (ED)’ }

Actions = { ‘ST’: [‘MI’], ‘MI’: [‘CA’, ‘IA’], ‘CA’: [‘IA’, ‘MI’], ‘IA’: [‘CC’, ‘CA’], ‘CC’: [‘FP’, ‘IA’, ‘PP’], ‘FP’: [‘PP’, ‘CC’], ‘PP’: [‘PR’, ‘CC’, ‘IA’], ‘PR’: [‘CM’, ‘PP’, ‘FP’], ‘CM’: [‘BI’, ‘IA’], ‘BI’: [‘CM’, ‘ED’, ‘FP’], ‘ED’: [‘ED’], ‘ELSE’: -inf }

For each state, possible transitions to other states are listed as valid actions. At last, we include a provision in our framework to inform the LLM that the remaining transitions are invalid, using a special notation at the end of the list of actions: **‘ELSE’: -inf**. This approach allows the Q-Learning algorithm to model invalid transitions with a infinitely large negative reward. As a result, in order to maximize cumulative reward, our framework, in principle, avoids these actions. For legal matter intake workflow, **Rewards**, **Requirements**, and the details of episode simulation closely follow the discussion mentioned for research scientist workflow, as such for brevity, we avoid repeating them here.

6 Evaluation

We experimented with our prompting framework with the GPT-4 model (knowledge cutoff: April 2023) for both case studies which includes four configurations, and for 10 iterations per configuration. In cases that the GPT model initially fails to deliver the correct answer, we re-prompt it with “*Did your output satisfy all of the following requirements? If not, you MUST take a fresh approach and execute it if necessary.*”, and we repeat the states, actions, rewards, and requirements (refer to Figure 5 in the appendices), with a maximum of 5 iterations, $Max_Iter = 5$. Table 1 summarizes the distribution for the number of iterations and the expected reward for the optimal workflow for both research scientist and legal matter intake. We also test two γ (discount factor) configurations. We first do not specify (UNS) the γ value and allow the LLM to decide its own value which can be variable on each iteration. In the second configuration, we provide a fix value for γ as a part of the task requirements and then measure the variability of the calculated optimal reward for both cases. Overall, we run the experiments for 40 iterations, 20 iterations for each use case.

6.1 Research Scientist

In 19 out of 20 iterations for this use case, our approach resulted in the following optimal workflow for the research scientist: *Start (ST) → Initiate Research (IR) → Literature Review (LR) → Manuscript Drafting (MD) → Submission to Venue (SV) → Peer Review (PV) → Result Publication (RP) → End (ED)*. This sequence is optimal for this use case and bypasses common workflow steps such as experiment planning and execution. This outcome confirms that the LLM is not outputting a commonly followed flow based on its embedded knowledge if not optimal. The optimal reward considering $\gamma = 0.9$ is -4.7 . Variations in the LLM’s calculation of the expected reward for UNS configuration are generally attributed to the use of different γ values, and we observe that setting γ value as part of the input requirements results in zero reward variations.

6.2 Legal Matter Intake

Similarly, for legal matter intake workflow, in all 20 iterations, the LLM was able to reach the optimal flow: *Start (ST) → Matter Intake (MI) → Initial Assessment (IA) → Client Communication*

(CC) → Proposal Preparation (PP) → Proposal Review (PR) → Case Management (CM) → Billing (BI) → End (ED). The optimal reward considering $\gamma = 0.9$ is -5.2 , and similar to the first use case, setting the γ value as part of workflow requirements results in zero variance in optimal values.

Task	(γ)	Iterations $\mathcal{N}(\mu, \sigma)$	Optimal Reward $\mathcal{N}(\mu, \sigma)$
Research Scientist	UNS 0.9	(1.6, 0.5) (2.0, 0.5)	($-4.3, 1.8$) ($-4.7, 0.0$)
Legal Matter Intake	UNS 0.9	(1.8, 0.4) (1.5, 0.5)	($-4.7, 2.0$) ($-5.2, 0.0$)

Table 1: Results for Research Scientist and Legal Matter Intake workflows. UNS: Unspecified γ value.

For both use cases, on average, our framework was able to reach the optimal workflows with ≤ 2 iterations, as illustrated on “**Iterations**” distribution in Table 1. Our observation was that in most cases, LLM was able to satisfy most of the requirements in solving the task optimization in the first prompt, i.e., Figure 4. The second prompt, i.e., Figure 5, then allowed the LLM to revisit the requirements and fix if some of the conditions are missed in the first trial. Overall, our approach demonstrated the effectiveness of iterative prompting in order to ensure that LLM is navigated towards satisfying all task optimization requirements.

7 Discussion

7.1 RL Task Complexity

Although our provided use cases might seem limited, we believe as the available prompt context of LLMs is increasing over 100K tokens (OpenAI, 2023b; Anthropic, 2023), the potential to handle more complex RL problems using our proposed framework is growing. Additionally, with the recent announcement of Custom GPTs (OpenAI, 2023a), we anticipate that LLMs that are tailored to specific user preferences and requirements will further allow us to optimize according to our RL framework. Therefore, our approach can be integrated as a custom NLP module that customizes an LLM for optimizing RL tasks, while allowing the end user to interact with LLM-based agent through natural language.

7.2 User Logs

In our research scientist and legal matter intake examples, we demonstrated how a workflow can be optimized as an RL problem. This approach can

be applied to workflow logs from various personnel in an enterprise setting instead of the simulated workflows that we generated with the LLM. By analyzing these workflow logs and targeting metrics to optimize, such as task efficiency, we can optimize the workflows of employees, and thus improve overall enterprise efficiency. Additionally, we initially relied on the LLM to generate simulations and episodes of interactions with provided states, actions, and rewards. Compared to the LLM simulations, the available user log data can better represent the actions and rewards for different states and result in a more accurate implementation of our MDP framework. This aspect serves as an interesting avenue for our future research.

7.3 LLM-based Planning

In our work, we primarily aimed to describe the plan of solving the use case for LLM to follow. An extension of our work to consider is to perform this step with the LLM as well (Valmeekam et al., 2023). Such that we first ask the LLM to generate several plans to solve the use cases and list down the steps involved in each. We then navigate the LLM through the steps for each plan via our prompting platform and collect the outcomes of different plans. Lastly, the LLM can be also serve as a judge to select the best plan based on the outcomes of different plans.

Furthermore, every step of the planning can also be delegated to specialized multimodal LLM agents (MLLMs) (Yin et al., 2023). To provide a concrete example, we dive into the "Conflict Assessment" in the legal matter intake flow as illustrate in Figure 3. Assume the agent needs to check for conflicts through an interactive application called National Conflict Lookup Service (NCLS). The information captured during the "Matter Intake" state includes the client's full name along with case-related details and is available for use by the "Conflict Assessment" MLLM. The specialized MLLM has learned how to successfully perceive the NCLS application and act on it by entering the client's full name into the appropriate text field, pressing the "Check for conflicts" button, and resulting in either a "No conflict found" or "Conflict found" status update, so it can transition to the next ideal workflow state.

The left box on Figure 3 provides a sketch on how the MLLM agent can learn to complete the task through interactions with the NCLS environment, similar to ScreenAgent (Niu et al., 2024).

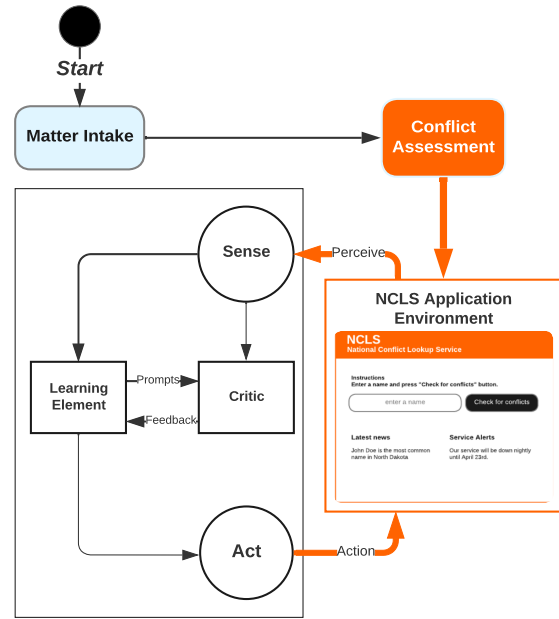


Figure 3: MLLM agent for Conflict Assessment state.

Upon entering the "Conflict Assessment" state, and with an interactive RL-based problem-solving approach in mind, the MLLM agent observes the state's default prompt for the task, e.g., "Check client's name for conflict". Assuming a first pass at performing the task, the *Learning Element Acts* on the NCLS and *Senses* the observations. The *Critic* provides feedback on the agent's actions and allow it to refine its approach to performing the task. In essence, the MLLM agent will learn and iteratively rephrase the current state's prompt in "Check for conflicts". We pursue MLLM-based planning and task optimization as directions for our future work.

8 Conclusion

With the advent of Large Language Models (LLMs), recent research has aimed to build on their reasoning and problem-solving capabilities and adapt them to different complex tasks. In this study, we introduced a novel approach to frame reinforcement learning problems as LLM prompting tasks as a novel way of solving RL problems through the user and LLM interactions. We developed an LLM prompting methodology that represents different elements of an RL problem based on the Markov Decision Process. Additionally, we leveraged LLMs for episode simulation and Q-Learning optimization and derived the optimal policy and reward based on our iterative prompting framework. In future work, we plan to expand our work to additional tasks and optimize workflow processes in the enterprise environment with LLM-based planning.

9 Limitations

Scope. Although we aimed to introduce a generic approach for RL problem formulation, a few limited cases are experimented with. Thus, while we think our prompting technique should be generalizable to other RL tasks, further experimentation to verify broader evidence is required.

LLMs Variability. Different LLMs may have noticeable differences in their performance on our illustrated use cases, and thus, can impact our findings. In addition, in our use cases, we also observe that LLM outputs for different iterations can be variable and it can often be difficult to reproduce the same exact results through prompting. For this reason, we conducted our study on a state-of-the-art (SOTA) LLM to establish the initial viability of our framework. We believe that as the LLMs continue to improve and as SOTA advances, this only further validates the feasibility of expanding our RL framework to tackle more complex RL or planning problems. Furthermore, to achieve more consistent LLM outputs and structured data formats, approaches such as LLM Functions (OpenAI, 2023b) can be leveraged.

Ethical Considerations. As we use LLMs as the basis of our work, ethical considerations evolve on the appropriateness of the generated outputs (Head et al., 2023). We reason that the chance of harmful outputs for our application should be minimal as our framework aligns the prompts to specific RL tasks. Furthermore, when analyzing real user workflow logs, it is imperative to implement steps to ensure privacy.

References

- Anthropic. 2023. Claude 2.1. <https://www.anthropic.com/index/claude-2-1>. Accessed: December 08, 2023.
- Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michal Podstawski, Hubert Niewiadomski, Piotr Nyczyk, et al. 2023. Graph of thoughts: Solving elaborate problems with large language models. *arXiv preprint arXiv:2308.09687*.
- Yuhao Dan, Zhikai Lei, Yiyang Gu, Yong Li, Jianghao Yin, Jiaju Lin, Linhao Ye, Zhiyan Tie, Yougen Zhou, Yilei Wang, et al. 2023. Educhat: A large-scale language model-based chatbot system for intelligent education. *arXiv preprint arXiv:2308.02773*.
- Ali Hakimi Parizi, Yuyang Liu, Prudhvi Nokku, Sina Gholamian, and David Emerson. 2023. A comparative study of prompting strategies for legal text classification. In *Proceedings of the Natural Language Processing Workshop 2023*, pages 258–265, Singapore. Association for Computational Linguistics.
- Cari Beth Head, Paul Jasper, Matthew McConnachie, Linda Raftree, and Grace Higdon. 2023. Large language model applications for evaluation: Opportunities and ethical implications. *New Directions for Evaluation*, 2023(178-179):33–46.
- Bin Hu, Chenyang Zhao, Pu Zhang, Zihao Zhou, Yuanhang Yang, Zenglin Xu, and Bin Liu. 2023. Enabling intelligent interactions between an agent and an llm: A reinforcement learning approach.
- Michael Janner, Qiyang Li, and Sergey Levine. 2021. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34:1273–1286.
- Inwon Kang, Sikai Ruan, Tyler Ho, Jui-Chien Lin, Farhad Mohsin, Oshani Seneviratne, and Lirong Xia. 2023. Llm-augmented preference learning from natural language. *arXiv preprint arXiv:2310.08523*.
- Valentin Liévin, Christoffer Egeberg Hother, and Ole Winther. 2022. Can large language models reason about medical questions? *arXiv preprint arXiv:2207.08143*.
- Pongsakorn Limna, Tanpat Kraiwanit, Kris Jangjarat, Prapasiri Klayklung, and Piyawatjana Chocksathaporn. 2023. The use of chatgpt in the digital era: Perspectives on chatbot implementation. *Journal of Applied Learning and Teaching*, 6(1).
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.
- Runliang Niu, Jindong Li, Shiqi Wang, Yali Fu, Xiyu Hu, Xueyuan Leng, He Kong, Yi Chang, and Qi Wang. 2024. Screenagent: A vision language model-driven computer control agent. *arXiv preprint arXiv:2402.07945*.
- OpenAI. 2023a. Chatgpt plugins. <https://openai.com/blog/chatgpt-plugins>. Accessed: December 10, 2023.
- OpenAI. 2023b. Function calling - openai api. <https://platform.openai.com/docs/guides/function-calling>. Accessed: 2024-02-29.
- OpenAI. 2023. Gpt-4 technical report.
- OpenAI. 2023a. Introducing GPTs. <https://openai.com/blog/introducing-gpts/>. Accessed: December 08, 2023.

- OpenAI. 2023b. New models and developer products announced at devday. <https://openai.com/blog/new-models-and-developer-products-announced-at-devday/>. Accessed: December 08, 2023.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.
- Martin L Puterman. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.
- Noah Shinn, Beck Labash, and Ashwin Gopinath. 2023. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*.
- Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems extended abstracts*, pages 1–7.
- Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. 2023. [On the planning abilities of large language models - a critical investigation](#). In *Thirty-seventh Conference on Neural Information Processing Systems*.
- Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2023. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.
- Mark Woodward, Chelsea Finn, and Karol Hausman. 2020. Learning to interactively learn and assist. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 2535–2543.
- Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*.
- Zelai Xu, Chao Yu, Fei Fang, Yu Wang, and Yi Wu. 2023. Language agents with reinforcement learning for strategic play in the werewolf game. *arXiv preprint arXiv:2310.18940*.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.
- Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. 2023. A survey on multimodal large language models. *arXiv preprint arXiv:2306.13549*.
- Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

A Prompts

Figures 4 and 5 compile all the prompts used for our demonstrated use cases. Because of the generalizability in their designs, these prompts can readily serve as a template for formulating additional RL-based use cases. In addition to leveraging self-reflection (Shinn et al., 2023), we also implemented capitalization of critical instructions to emphasize the output requirements and ensure more consistent generations from the LLM (Kang et al., 2023). The aforementioned prompting techniques, in addition to the code generation capabilities of GPT (OpenAI, 2023a), allow the LLM to accurately implement and simulate the MDP based on the provided input and thereby produce the optimal policy.

Problem Context: You are an RL agent tasked with maximizing cumulative reward for a given task.

You will be provided with the task, states, possible actions at each state, and rewards.

Task: Workflow of a research scientist.

States = { 'Start (ST)', 'Initiate Research (IR)', 'Literature Review (LR)', 'Experiment Plan (EP)', 'Experiment Execution (EE)', 'Data Analysis (DA)', 'Manuscript Drafting (MD)', 'Submission to Venue (SV)', 'Revision of Manuscript (RM)', 'Peer Review (PR)', 'Result Publication (RP)', 'End (ED)' }

Actions = { 'ST': ['IR'], 'IR': ['LR', 'EP'], 'LR': ['EP', 'MD'], 'EP': ['EE'], 'EE': ['DA'], 'DA': ['MD', 'EP'], 'MD': ['SV', 'RM'], 'SV': ['RM', 'PR'], 'RM': ['EE', 'SV'], 'PR': ['RM', 'RP'], 'RP': ['ED'], 'ED': ['ED'], 'ELSE': -inf }

Rewards = { 'ED': 0, else: -1 }

Requirements:

1. Solve it with Q-learning. $\gamma=0.9$
2. Transitions from one state to another are only allowed based on provided 'Actions'. Other actions are not possible.
3. First, simulate the environment for 1000 episodes and fill in the Q-table for states. If an episode goes more than 100 steps terminate that episode.
4. Episodes MUST begin at "Start" and finish at "End".

Output:

Print the Q-table, and list the state/action pairs and their "Q-values" for the optimal episode from "Start" to "End".

Figure 4: This prompt is used to model the research scientist workflow based on the Markov Decision Process.

Did your output satisfy all of the following requirements? If not, you MUST take a fresh approach and execute it if necessary.

Task: Workflow of a research scientist.

States: $\{s_1, s_2, \dots, s_n\}$

Actions: $\{a_1, a_2, \dots, a_m\}$

Rewards: $\{r_1, r_2, \dots, r_k\}$

1. Solve it with Q-Learning. gamma=0.9
2. Transitions from one state to another are only allowed based on provided 'Actions'. Transitions from one state to another are only allowed based on possible actions.
3. First, simulate the environment for 1000 episodes and fill in the Q-table for states. If an episode goes more than 100 steps terminate that episode.
4. Episodes MUST begin at "Start" and finish at "End".
5. Print the state/action pairs and their "Q-values" for the optimal episode from "Start" to "End".

Figure 5: This prompt is used for iterative verification of the LLM outputs until reaching the desired outputs. **Task**, **States**, **Actions**, and **Rewards** are place-holders for use case specific values, e.g., values from Figure 4 can be used here for research scientist workflow.