METAREFLECTION: Learning Instructions for Language Agents using Past Reflections

Priyanshu Gupta*, **Shashank Kirtania***, **Ananya Singha***,
Sumit Gulwani, Arjun Radhakrishna, Sherry Shi, Gustavo Soares
Microsoft
{priyansgupta, t-skirtania, ananyasingha, sumitg, arradha, shersh, gustavo.soares}@microsoft.com

Abstract

The popularity of Large Language Models (LLMs) have unleashed a new age of *Language Agents* for solving a diverse range of tasks. While contemporary frontier LLMs are capable enough to power reasonably good Language agents, the closed-API model makes it hard to improve in cases they perform suboptimally. To address this, recent works have explored ways to improve their performance using techniques like self-reflection and prompt optimization. Unfortunately, techniques like self-reflection can be used only in an online setup, while contemporary prompt optimization techniques are designed and tested to work on simple tasks. To this end, we introduce METAREFLECTION, a novel offline reinforcement learning technique that enhances the performance of Language Agents by augmenting a *semantic memory* based on experiential learnings from past trials. We demonstrate the efficacy of METAREFLECTION by evaluating across multiple domains, including complex logical reasoning, biomedical semantic similarity, open world question answering, and vulnerability threat detection, in Infrastructure-as-Code, spanning different agent designs. METAREFLECTION boosts Language agents' performance by 4 % to 16.82 % over the raw GPT-4 baseline and performs on par with existing state-of-the-art prompt optimization techniques while requiring fewer LLM calls. We release our experimental code at: aka.ms/metareflection-code

1 Introduction

Large Language Models (LLMs), such as GPT-4 (OpenAI, 2023), have gained significant popularity in recent years due to their ability to generate human-like text and solve complex tasks across various domains. To leverage these models, users typically craft prompts with instructions that are tailored to a specific task. Furthermore, many practical LLM applications setup complex multi-step systems with multiple LLM calls chained together (LangChain, 2023) or LLM calls with different prompts called in succession (Wu et al., 2023). Given their close resemblance to reinforcement learning agents, it has become useful to model these systems as Language Agents (Wu et al., 2023)

Building up on these insights, recent works (Shinn et al., 2023; Madaan et al., 2023) have showed that the performance of such language agents can be improved using *verbal reinforcement learning* across multiple conversational turns, by providing feedback at the end of a failing trajectory and asking the agent to reflect on its mistakes. The reflective text is then stored as episodic memory to improve future trajectories on the same task.

Parallel to this, there has been recent advancements in developing prompt optimization techniques (Pryzant et al., 2023; Wang et al., 2023). These techniques typically start with a seed human prompt and a train dataset of Input-Output pairs. Leveraging error feedback over failing cases, they deploy various search strategies to find prompt variants that can give the optimal results on train set.

While prompt optimization techniques exist for single LLM call setups, they are not designed for complex multi-step language agents. Here, techniques like self-reflection can help improve a language agent's performance significantly. However, these are online reinforcement processes that depend on the availability of performing multiple turns with a feedback mechanism on the same task instance and aren't applicable to similar new tasks.

Motivated by the *semantic memory* observed in humans (McRae and Jones, 2013; Weng, 2023; Paischer et al., 2023) we introduce METAREFLECTION, an offline reinforcement learning technique that works by augmenting a *semantic memory* to represent experiential learnings from trials in an of-

*Equal Contribution

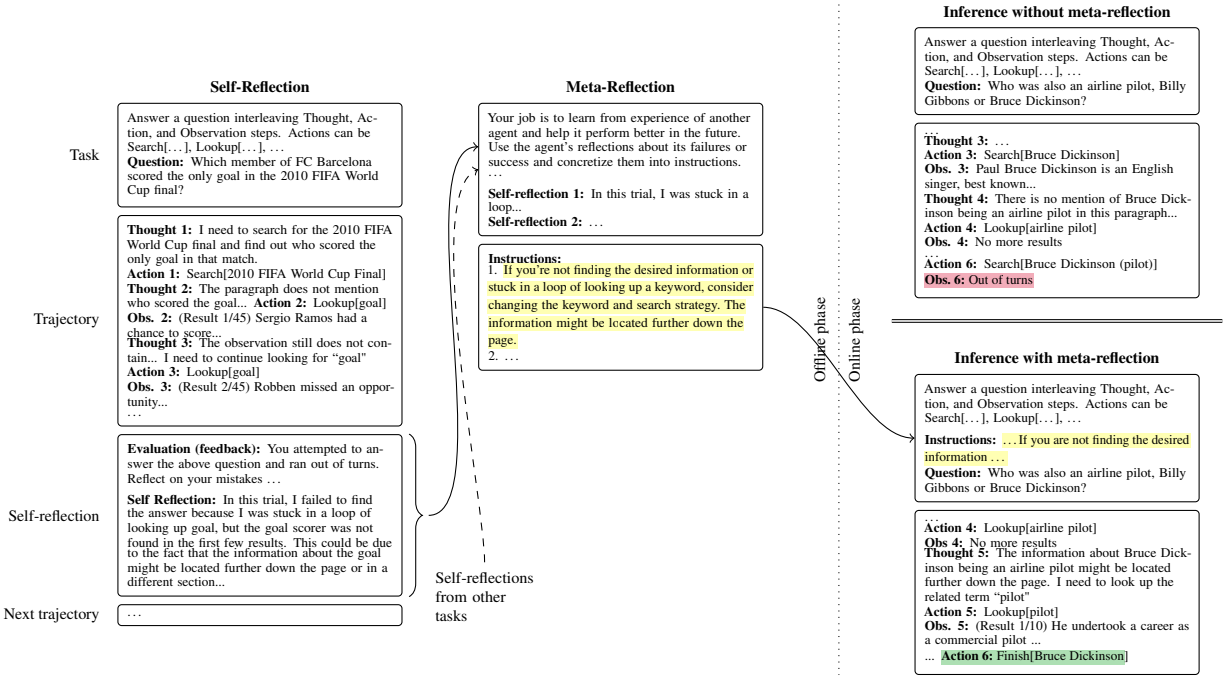

Figure 1: METAREFLECTION in the HOTPOTQA domain with a REACT agent

fline training phase. At a high level, METAREFLECTION works by simulating various trials with samples from a training dataset, gather self-reflections from failing trials and iteratively builds an experiential *semantic memory* by generalizing the self-reflections into 'meta-reflections' that takes the form of instructions to the language agent.

We evaluate the METAREFLECTION technique across multiple scenarios: 3 complex logical reasoning datasets (Suzgun et al., 2022), a biomedical semantic similarity dataset (Soğancıoğlu et al., 2017), an open world question answering (Yang et al., 2018), and a new vulnerability threat detection in a new Infrastructure-as-Code (IAC) dataset.

The IAC vulnerability detection dataset, in particular, is a new work that we are introducing with this paper, entailing detection of 10 unique common security vulnerabilities found in Terraform (HashiCorp, 2023) cloud infrastructure configuration files. While static analysis tools like Terrascan (Tenable, 2023) exist, they are not able to detect semantic nuances beyond the standard hard-coded detection patterns. Being a low-resource language, vanilla GPT-4 struggles to effectively detect the vulnerabilities. This makes it a particularly interesting dataset for domain-specific reasoning.

METAREFLECTION achieves 4% to 16.82% higher accuracy over raw GPT-4 baseline. For setups, where they are directly comparable, METAREFLECTION performs on par or better than state of the art prompt optimization techniques, while requiring fewer LLM calls for learning. We also demonstrate how METAREFLECTION can be applied to multi-step agents for offline learning.

To summarize, we make the following contributions:

- We present METAREFLECTION, a technique for learning *semantic memory* for language agents using offline training simulations. To the best of our knowledge, this is the first approach towards learning semantic memory for generic language agents.
- We introduce a new dataset for vulnerability threat detection in IAC. The dataset comprises 416 challenging vulnerability detection tasks.
- We conducted an extensive evaluation of the METAREFLECTION technique across multiple distinct domains: vulnerability threat detection, causal reasoning, domain-specific semantic similarity, and open-domain question answering, demonstrating strong improvements in agent performances. (Section 3).

2 METAREFLECTION

We consider a setup of an LLM Agent A interacting with an environment Env to solve a task T, characterized by the start state S_0. At time step t, the Agent A receives an observation o_t at state S_t and

Algorithm 1 METAREFLECTION

Require: Train data D_T, Val. data D_V, Agent A
Require: Environment Env
Ensure: Meta-reflection instructions Instr

```
1: Instr ← ⊥
2: for all b ∈ Batch(D_T) do
3:     v ← RandomSample(D_V)
4:     for all Trials ∈ {0, 1 ... MaxTrials} do
                                        ▷ Iterative Refinement
5:         Initialize A with Instr
6:         trajectories ← {A.Run(i) | i ∈ b }
7:         fT ← Env.GetFailingTrajectories(trajectories)
8:         if len(fT) == 0 then
9:             break                    ▷ Early Stopping
10:        refl ← { A.SelfReflect(t) | t ∈ fT }
11:        Instr* ← A.MetaReflect(Instr, b, refl)
12:        if A.ShowsImprovement(Instr*, Instr, b, v, Env)
   then                                 ▷ Validation
13:            Instr ← Instr*           ▷ Else Backtrack
14: return Instr
```

samples an action $\mathsf{a_t}$ using a policy $\pi(\mathsf{a}_t|\mathsf{T}_t, \mathsf{P}_t)$ where $\mathsf{T}_t = \{\mathsf{o_0}, \mathsf{a_0}, \mathsf{o_1} \ldots, \mathsf{a_t}, \mathsf{o_t}\}$, represents the trajectory of the agent and the prompt $\mathsf{P_t}$ is the prompt that drives the action selection at time step t. Eventually, the agent reaches a state S_n where it receives a reward from the agent $\mathsf{r} = \mathsf{R}(\mathsf{S_n})$. This state marks the correct/incorrect completion of the task T by the language agent.

METAREFLECTION aims to improve the performance of such language agents by augmenting them with a *semantic memory* (McRae and Jones, 2013) in the form of rule based instructions **Instr** to store the experiential learnings, replacing $\pi(\mathsf{a_t}|\mathsf{T}_t, \mathsf{P})$ with $\pi(\mathsf{a_t}|\mathsf{T}_t, \mathsf{P} \bigoplus \mathsf{Instr})$. Algorithm 1 shows the outline of the METAREFLECTION process. At its core, the algorithm works by starting with an empty set of instructions and iteratively improving the instructions using small training batches.

For instance, if the agent $\mathsf{A_{react}}$ is a REACT-style agent working on a Question Answering Task over Wikipedia, at state S_0, the agent will be provided by a question Q. Now, the agent can choose to take a search action, upon which it will receive an observation o. The agent can then either chose to pick further search or lookup actions, or pick an answer action to return an answer. This action will mark a transition to the state $\mathsf{S_n}$ and a binary reward can be generated depending on the correctness of the answer. An even simpler example can be an agent $\mathsf{A_s}$ which given a Multiple-Choice Question, picks an answer and receives a positive reward if the answer is correct.

Agent, Environment and Self-reflection method METAREFLECTION procedure uses the following components at its core:

(a) an agent A (i.e., an RL actor) that is based on a language model, (b) an environment Env that generates rewards for specific actions, given a agent's trajectory, and (c) A SelfReflect method on the lines of (Shinn et al., 2023) that produces a verbal reinforcement given an agent's trajectory (line 10, Algorithm 1) This is in contrast to prior state of the art prompt optimization techniques (Wang et al., 2023; Pryzant et al., 2023), where optimization relies solely on the final action of the agent and cannot be applied on the intermediate un-observed states of the actor. Additionally, we adapt the agent A to be parameterized by a set of instructions in addition to the standard task description and agent behaviour prompt. In our implementation and experiments, we utilize either a single-step agent (a vanilla single-shot Language Model agent) or a multi-step agent based on REACT (Yao et al., 2023c) and COT (Wei et al., 2023). While METAREFLECTION makes no distinction between the two, we make this distinction to ease comparison with other prompt optimization techniques, which only work in the single-step agent configuration. More information on agent configurations can be found in Section 3.3.

The MetaReflect step. The MetaReflect method is designed to take as input a prior set of instructions Instr, a set of self-reflections refl, the training data $\mathsf{D_T}$ and a validation data $\mathsf{D_V}$ and will produce an updated set of instructions Instr^* as denoted in line 11 of Algorithm 1.

For the MetaReflect method, we use a standard language model with a prompt that instructs the LLM to observe the reflections, the training data, and produce a new non-case specific instruction style memory. Further, the prior memory is also passed as input so that the output is a generalization of the prior learnings. In our implementation, this meta-reflection and generalization are done in the same prompt for efficiency. Alternatively, new instructions can be generated first and then combined with existing ones.

We specify that the instructions need to take the form of a list. Hence, the meta-reflection step in line 11 typically either (a) updates the list by adding a new item, or (b) combines one or more previous items with learnings from the self-reflections to produce a shorter list. For example, one

meta-reflection instruction learned during our HOTPOTQA experiments suggested including the profession when searching for a person to narrow down results. In a subsequent batch, the self-reflection step produces a reflection that mentions adding search terms like release date when searching for movies. The MetaReflect step may combine the previous instructions with the current self-reflections either by appending a new item to the list clarifying the strategy to search for movies, or may generalize the previous item to something like "*When searching for specific entities, use additional contextual information to augment the primary search terms with secondary keywords corresponding to the characteristics of the entity*".

Validation and Backtracking In each iteration, after MetaReflect, we validate the quality of the new instructions. Due to the sparse reward signals leading to poor self-reflections or over-generalization of the meta-reflection instructions, we may end up with instructions that are of a poorer quality than the prior instructions. The poorer instructions may also be due to general capricious, unpredictable nature of large language models. Therefore, we validate the new instructions by testing them on training data and a random sample of the validation set to ensure that they perform better than the prior instructions as depicted in line 12 of Algorithm 1. Ideally, we would do this validation over the full validation dataset. However, in our case, we only validate on the current batch to balance quality of instructions and efficiency. In cases where the updated instructions perform poorly compared to the prior ones, we *backtrack* to prior instructions as depicted in line 13.

As an example, in the previous paragraph the meta-reflection step replaced the specific instruction on how to search for persons with a more general instruction on how to search for entities. However, it is possible that these general instructions are too vague (especially for smaller, less capable models) and the client agent is not able to apply them correctly to the case of searching for persons.

Iterative Refinement Motivated by the self refining behaviour of Language models as demonstrated in (Shinn et al., 2023), we use multiple attempts at meta-reflection for each batch (line 4 through line 13) until we see no failures in the current batch or exhaust a maximum number of trials (set to 3 for all our experiments). Similar to how self-reflections help optimize Language agents' trajectories towards achieving a task, the feedback on failed trajectories from the Instr* act as implicit *verbal reinforcements* in the meta-reflection process. These *verbal reinforcements* can then be used by the meta-reflection step to guide the instruction search. This trajectory driven iterative refinement strategy reduces the chances of repeating mistakes, such as proposing incorrect refinements that were already tried in the past, during the instruction refinement process.

In the single step agent setups where they are directly comparable, we observe that this iterative refinement strategy leads us to similar performance as state of the art prompt optimization techniques, with lesser number of LLM calls.

3 Experimental Setup

3.1 Datasets

We evaluate METAREFLECTION on datasets from different domains like vulnerability threat detection (IAC), question answering (HOTPOTQA), Complex Reasoning (BIGBENCH), Biomedical Semantic Similarity (BIOSSES).

Vulnerability Threat Detection (IAC) Infrastructure-as-Code (IAC) is a popular method of configuring cloud infrastructures, on platforms such as Azure and AWS, using a configuration coding language. Here, we focus on Terraform, a leading IAC platform by Hashicorp (HashiCorp, 2023), as well as Azure, the cloud computing platform by Microsoft, which comes as a reusable configuration component. Such Cloud infrastructures are prone to security vulnerabilities such as open ports and exposed administrator accounts (Tenable, 2023). Vulnerability detection via static analysis such as Terrascan (Tenable, 2023) of IAC files is a hard problem due to the expressivity and complexity of the configuration language and the diversity of the resources being handled across multiple infrastructure providers (e.g., Amazon AWS and Microsoft Azure).

This opens up the possibility of using an LLM to perform vulnerability detection which entails checking if a given Terraform module violates a given Terrascan policy. To evaluate the efficacy of METAREFLECTION on vulnerability detection task we collected 202 Terraform modules by mining GitHub repositories and post processing it further to achieve 186 data points which is then split 40 : 60 for train and test respectively. You can find more

Dataset	Train set	Test set
BIOSSES	60	40
CASUAL JUDGEMENT	90	100
EPISTEMIC REASONING	500	500
TEMPORAL SEQUENCE	300	500
IAC Vulnerability Detection	166	250
HOTPOTQA	50	80

Table 1: Count of train and test set distribution across benchmarks

information on IAC, Terraform, Terrascan and data collection in Appendix A.2

Complex Reasoning (BIGBENCH) Big-Bench Hard (BBH) (Suzgun et al., 2022) consists of a subset of particularly challenging tasks from BIG-Bench (Srivastava et al., 2022) that contains challenging reasoning questions. In this work, we pick 3 datasets from BBH - 1. **Causal Judgement** 2. **Temporal Sequence** 3. **Epistemic Reasoning** and follow the exact same test and train distribution used in (Wang et al., 2023). Table 1 shows the test and the train distribution for the benchmarks.

Biomedical Semantic Similarity (BIOSSES) is a biomedical sentence similarly dataset (Soğancıoğlu et al., 2017). Each instance in the dataset comprises of two sentences which are to be compared. As prior work (Wang et al., 2023), we model the problem as a classification task between labels 'similar', 'non-similar' or 'somewhat similar' and utilize the exact same test and train data splits.

Question Answering (HOTPOTQA) The dataset (Yang et al., 2018) is a Question-Answering dataset consisting of $113k$ question-and-answer pairs over Wikipedia. A typical system working over this dataset first performs a retrieval over wikipedia and the reasons over the retrievals to come up with an answer. Besides the retrieve and reason setup, the dataset also comes in 2 resoning only settings: 1. **GT**: where each Question is accompanied with the most relevant supporting *ground truth* documents and 2. **Distractor** where the Question is accompanied with the ground truth documents, alongside some distracting documents making context analysis and interpretation more challenging. Given the large scale of the dataset, we adversarially select (Appendix A.3) test samples from the test split of the dataset to ensure good sample diversity.To perform adversarial sampling, we identify samples where the Agent **A** consistently fails and conduct up to three self-reflection trials to correct the response. If the Agent **A** still fails, we discard these samples. This method ensures we gather challenging examples while filtering out noisy ones, and we sample 40 and 80 examples for the REACT train and test sets, and 50 and 80 for COT settings, respectively.

3.2 Baselines

In absence of techniques for direct comparison, we compare METAREFLECTION against strong contemporary prompt optimization baselines 1. PROTEGI and 2. PROMPTAGENT across multiple single step agent settings.

PROTEGI (Pryzant et al., 2023) performs prompt optimization by leveraging batch-wise error feedback as textual gradients and use them to generate multiple variations of the current prompt. Notably, these textual gradients are prompt update 'guidelines' generated by looking at the failing error output. They then deploy beam search to find the best prompt by iteratively evaluating and generating more prompt candidates.

PROMPTAGENT (Wang et al., 2023) Similar to PROTEGI, PROMPTAGENT also leverages error feedback to generate prompt candidates. They further optimize prompt search using a principled Monte-Carlo Tree Search and identify high-reward paths to find the best prompts.

3.3 Agent Configurations

The generic Language Agent A defined in Section 2 can operate upon multiple time steps before reaching a response state $\mathbf{S_n}$. While, the METAREFLECTION algorithm is broadly applicable to all such agents, for the purpose of simplification of comparison, we distinguish our agent setups in 2 categories:1. *Single-Step* and 2. *Multi-Step* Agents

Single-Step Agent We call an Agent A a Single-Step Agent if it takes exactly 1 time step starting from the initial state S_0 to reach an answer state S_1, without traversing intermediary belief states. In practice, what it means is that the agent is characterized by a single zero-shot prompt that always elicits the agent to generate a response solving the input task. We define such single step agents for the IAC, BIOSSES and the BIGBENCH datasets.

Multi-Step Agent We define Multi-Step Agents to be agents with atleast one intermediate belief state. In our evaluations we consider two well established multi-step agent style COT (Wei et al.,

2022) and REACT (Yao et al., 2022). The COT setup entails exactly 2 time steps - the first transition generates the 'thought' and the next transition generates the 'answer'. The REACT setup, however, can entail multiple time steps with 'actions' for context gathering, before generating an 'answer' action. For the purpose of our evaluation, similar to (Shinn et al., 2023), we use the different HOTPOTQA datasets to setup COT-(Distractor) and COT-(GT) for the Distractor and GT settings respectively and a REACT agent with search and lookup actions on the wikipedia corpus for the retrieval and reasoning setting.

3.4 Other Experimental Configurations

We use GPT-4-32k chat model as an LLM throughout the experiments, with a $temperature = 0$. For METAREFLECTION, we set the batch size to be 4 and maxRetries to be 3, and the random sample size for the validation set to be 5. To account for statistical variations, all the experiments are run for 3 runs and we report the averages and standard deviation for each of them.

4 Results

4.1 Comparison with the baselines

Table 2 and Table 3 compares METAREFLECTION with baseline GPT-4 and various prompt optimization baselines over multiple datasets in a single-step agent setting. Further, Table 2 also presents the average number of LLM calls it took by different systems to generate the optimal prompt.

We observe that METAREFLECTION fairs competitively with the other techniques, outperforming GPT-4 and PROTEGI over all the benchmarks. Compared to PROMPTAGENT we observe better performance in all the datasets except for EPISTEMIC REASONING where we see a slight regression. Notably, in all the configurations, METAREFLECTION required the least number of LLM calls for training (include any intermediate inference calls), with PROTEGI and PROMPTAGENT requiring upto 17 and 1.5 times more LLM calls. In general, we observe that even with a lower number of required LLM calls, METAREFLECTION performs at par with PROMPTAGENT for generic causal reasoning tasks, while outperforming it on highly contextual domain-specific settings like IAC and BIOSSES.

Qualitative evaluations reveal that the instructions learned using METAREFLECTION capture

Terraform Policy Description:

```
{
    "name": "reme_containerRegistryResourceLock",
    "resource_type": "azurerm_container_registry",
    "description": "Ensure Container Registry has locks",
    "reference_id": "accurics.azure.AKS.3",
    "category": "Resilience",
    "id": "AC_AZURE_0185"
}
```

ProTeGi optimized prompt:

Verify if module interacts with "azurerm_container_registry" type. Check for constructs in the code that ensure the Azure Container Registry has locks. In case there are constructs that provide these locks, classify the module as "Compliant" . . .

PromptAgent optimized prompt:

The policy is constructed around the 'Container Registry' resource type. Identify module as 'Compliant' if it meets these conditions: *The resource type specified in the policy is not included in the module.* And *module does contain the resource type.* Label module as 'Violating' if it falls short of complying with regulations . . .

MetaReflection learned instructions:

The policy in question is "reme_containerRegistryResourceLock" which pertains to the "azurerm_container_registry" resource type and requires locks be implemented for resilience. Look for the "azurerm_container_registry" resource. If the "azurerm_container_registry" resource is absent then policy is not violated. Check if "azurerm_container_registry" resource has implemented locks. This can be done by looking for the "azurerm_management_lock" resource in the Terraform code and checking if it is associated with the "azurerm_container_registry" resource. If the "azurerm_container_registry" resource has implemented locks, then policy is not violated else it is.
. . .

Figure 2: Snapshot of the optimized prompt learnt by the baselines and METAREFLECTION on IAC task

subtle domain specific nuances, that other techniques don't. For example, consider the METAREFLECTION instruction compared to the PROMPTAGENT and PROTEGI prompt for the policy $reme\_containerRegistryResourceLock$ (Figure 2). The learnt METAREFLECTION instructions captures details like `azurem_management_lock` being a indicator of locking behavior in terraform, whereas the other baselines do not capture many such subtleties. This behaviour may be attributed to the objective driven search strategy deployed in METAREFLECTION, centered around generalizing self reflections into semantic rules, compared to a more random Monte-Carlo search strategy deployed by other techniques.

4.2 METAREFLECTION for Multi-Step Agents

In the previous section we discussed how METAREFLECTION fairs in settings with a single-step agent setup. In this, section we will evaluate METAREFLECTION with multi-step agents.

Prompt optimization techniques like PROMPTAGENT and PROTEGI are designed to model systems with a single LLM call akin to the single-step

Dataset	GPT-4		ProTeGi			PromptAgent			MetaReflection		
	Acc	σ	Acc	σ	# calls	Acc	σ	# calls	Acc	σ	# calls
Biosses	70.00	3.17	80.75	0.71	6478	80	1.34	503	**84**	0.57	**475**
Causal Judgement	74	1.15	72.20	3.67	9725	73.40	3.19	351	**77**	2.00	**313**
Epistemic Reasoning	70.8	1.03	85.40	2.40	16640	**91**	0.83	2143	88.4	1.02	**1954**
Temporal Sequence	98.0	0.57	97.5	0.27	13824	98.3	1.11	2017	**99.1**	0.63	**1491**
IaC Vulnerability Detection	73.36	0.01	72.64	4.76	8148	84.17	6.91	354	**90.18**	0.28	**166**

Table 2: Comparison of MetaReflection with other baselines in a single-step agent set-up.# `calls` refers to average numbers of LLM calls required to generate optimal prompt

Policy	Data statistics			GPT-4	ProTeGi	PromptAgent	MetaReflection
	Files	Violating	Compliant				
networkPolicyEnabled	23	10	13	48	62.18	71.06	**83.34**
kubeDashboardDisabled	29	3	26	89	86.22	90.80	**94.63**
reme_keyVaultAuditLoggingEnabled	44	10	34	55	77.97	82.71	**98.18**
reme_containerRegistryResourceLock	35	9	26	53	68.37	90.22	**96.55**
reme_appGatewayWAFEnabled	59	36	23	72	68.06	88.70	**92.30**
networkPort22ExposedToInternetAz	27	4	23	94	87.84	89.02	**96.29**
networkPort22ExposedToPublicAz	35	6	29	94	85.22	80.60	**94.11**
reme_noSecurityGroupAssociated	90	60	30	70	67.20	82.60	**87.58**
reme_checkStorageContainerAccess	19	5	14	91	88.62	88.05	**100.00**
reme_resourceGroupLock	55	43	12	81	80.33	77.98	**82.90**

Table 3: Performance of MetaReflection and other baselines on IaC tasks using singe-step agent

agent setup. Generalizing them to multi-step agent scenarios present the following challenges:

1. How do we extend the notion of textual gradients to also include intermediate *belief* states in the absence of concrete feedback on them?

2. How do we ensure that the agent with the updated prompt continues to follow the agentic behaviour?

Given these challenges, to draw a valid comparison of these baseline against MetaReflection we perform a single step adaption of the original two step CoT. The adaption includes reducing the [Thought] and [Answer] steps within a single LLM call creating an analogue to single-step agentic flow and use this structure as a seed prompt.

Table 4 shows the performance of CoT style of agent on HotpotQA benchmark. We observe that MetaReflection outperforms other prompt-optimization baselines ProTeGi by 23.33% and PromptAgent by 3.25% for the GT setting. Similarly, we see a similar trend in the Distractor setting where MetaReflection technique outperforms ProTeGi by 24% and PromptAgent by 1.08%. Notably, compared to the best baseline PromptAgent, MetaReflection is capable of learning instructions to guide the Language Agent towards creating better thought actions instead of just optimizing it towards coming up with the right answer.

Towards truly multi-step agents While a CoT agent can be represented as a single prompt with one call for optimization, similar adjustment is not possible for all multi-step agents. To this end we consider the ReAct setup for HotpotQA unlike CoT which entails a strict two step process with thought action followed by an answer, HotpotQA ReAct setup can span over multiple steps of search and lookup actions. This leads to complex trajectories leading upto the final answer action for returning the answer. Besides, each action selection is conditioned on the previous action. Representing such a system with a single step prompt is non-trivial, making it hard to adapt existing prompt optimization techniques to work over such a set up.

MetaReflection procedure, on the other hand, allow us to seamlessly append **semantic memory** to such an agent and gather learning for each potential action states the agent can take. Table 5 shows the performance of multi-step ReAct style agent on HotpotQA. We observe that MetaReflection nearly doubles the performance of the baseline GPT4 agent.

For instance, in Figure 1, the updated *semantic*

Dataset	GPT-4		ProTeGi			PromptAgent			MetaReflection		
	Acc	σ	Acc	σ	# calls	Acc	σ	# calls	Acc	σ	# calls
HotpotQA (GT)	43.67	2.08	31.67	3.61	5403	51.75	1.52	383	**55.00**	1.0	303
HotpotQA (Distractor)	37.33	2.08	23.33	0.72	6132	46.25	2.16	568	**47.33**	1.15	314

Table 4: Comparison of MetaReflection with other baselines using CoT style agent on prompt refinement. *# calls* refers to average numbers of LLM calls required to generate optimal prompt.

	Acc.	σ
GPT-4	19.58	1.91
MetaReflection	**35.00**	1.25

Table 5: Comparison of METAREFLECTION and GPT-4 using ReAct style agent on prompt refinement for HotpotQA.

memory contain clear instructions for the ReAct agent to refine its search strategy if it gets stuck in loops. Note that this instruction guides the agent to better plan its trajectory rather than choosing the right answer. Later on, this learned rule aids the model in successfully concluding another trial where it was previously failing by explicitly guiding the action to look further down the context page to refine its lookup strategy, leading to the correct response, *Bruce Dickinson*. This is in contrast with the baseline attempt, where the agent ran out of trials by getting stuck in a loop.

Notably, the HotpotQA ReAct performance is lower than CoT due to the nature of the ReAct setup. The ReACT setting is based on the fullwiki setting where the task first involves "retrieving" relevant paragraphs from Wikipedia related to the Query and then "reasoning" over them. Being a more complex task, the ReACT setup exposes more failure surfaces (e.g., the system may not retrieve the correct information to begin with), which may lead to poorer performance compared to the simpler CoT setups.

5 Related Work

With the increasing ubiquity of black-box Large Language Models (OpenAI, 2023; Anil et al., 2023; Brown et al., 2020; Bai et al., 2022), there has been a growing interest in the community to develop strategies that can maximize the model's performance on a downstream task. These techniques may involve guiding an LLM to arrive at the correct answer (Wei et al., 2023; Zheng et al., 2023), output selection (Yao et al., 2023a; Poesia et al., 2022), picking up the right in-context examples (Khatry et al., 2023) or prompt optimization, etc. Being closely related to our work we dive deeper into contemporary Prompt optimization techniques in the literature.

5.1 Single Prompt Optimization.

Given the potentially infinite space of instructions, recent works have studied the problem of 'guided' prompt search instead. To this end, *OPRO* (Yang et al., 2023) proposes a prompt 'optimization' technique where prompt candidates coupled with eval-set metric evaluation act as 'few-shot' examples to generate new prompts.

APE (Zhou et al., 2023b) poses instruction generation as a synthesis problem and proposes techniques to effectively perform monte carlo search over the space of prompt candidates.

ProTeGi (Pryzant et al., 2023) and *PE2* (Ye et al., 2023) *Automated Prompt Engineering (APE)* The learned prompt can then be used during inference time in isolation. also leverage verbal feedback to generate and/or evolve task description prompts. In *PE2* they additionally, maintain an optimization history to iteratively improve the prompt. PromptAgent (Wang et al., 2023) also uses verbal feedback but views prompt optimization as a strategic planning problem and proposes a principled approach for prompt optimization.

Notably, these prompt optimization techniques are designed to work over single prompt single step-agents. The MetaReflection *semantic memory* learning can however be generalized for multi-step agents as well, as we demonstrated earlier. Even in the single-agent setup, MetaReflection differs from prompt optimization techniques in its focused objective-driven refinement of the memory instead of generating generic prompt candidates. This allows MetaReflection to perform especially good in knowledge intensive tasks.

5.2 LLMs as Agents

Recent works (Nakano et al., 2022; Schick et al., 2023) are leveraging these models to develop AI

agents that act as a controller, extending their perceptual and action capabilities through tool utilization (Yao et al., 2023b; Qin et al., 2023). LLM-based agents can demonstrate reasoning and planning skills at par to symbolic agents by employing techniques like CoT (Wei et al., 2023) and problem decomposition (Zhou et al., 2023a; Xi et al., 2023). Additionally, their ability to interact seamlessly using natural language comprehension allow them to operate in software development and research environment (Boiko et al., 2023; Qian et al., 2023). Furthermore, the interaction among multiple LLM-based agents can foster collaboration and competition, potentially leading to the emergence of complex social phenomena (Park et al., 2023).

5.3 Memory based methods in reinforcement learning

The concept of providing agents with controllable memory has a rich history. (Littman, 1993) discussed how hypothetical agents can utilize binary memory to store prior experiences, guiding their future action selection. In our context, we explore the interaction between language agents and their environment.

Previous research (Icarte et al., 2020) has demonstrated that memory-augmented agents can achieve globally optimal solutions. Inspired by the idea of rule-based semantic memory in humans(McRae and Jones, 2013), recent works have also investigated enabling reinforcement learning (RL) agents with semantic memory(Paischer et al., 2023).

6 Conclusion

In this work, we presented METAREFLECTION, a novel offline reinforcement learning technique that takes inspiration from how human brain store memories to enhance Language Agents by augmenting them with an experiential *semantic memory*. We further empirically demonstrated that the instructions learned using METAREFLECTION are more effective at capturing task-specific nuances. This behavior helps METAREFLECTION perform competitively with state of the art prompt optimization baseline for single-step agent scenarios. The objective driven iterative refinements also provide a significant advantage to capture the best learning at each iteration and save it to its semantic memory which helps the agent to perform better inference in next iteration. In the future, we would like to see how we can leverage our semantic memory in a multi-agent workflow. Sharing memories between agents can enhance synergy and improve task performance.

7 Limitation

METAREFLECTION currently relies on small scale held out validation for quantifying the efficacy of the batch. Such an approach leads to undesirable stochasticity in the results and cause instabilities in the learning. There is also scope of improving the quality of the reward signals that can potentially add to the stability of the learning.

References

Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernandez Abrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vlad Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, Guy Gur-Ari, Steven Hand, Hadi Hashemi, Le Hou, Joshua Howland, Andrea Hu, Jeffrey Hui, Jeremy Hurwitz, Michael Isard, Abe Ittycheriah, Matthew Jagielski, Wenhao Jia, Kathleen Kenealy, Maxim Krikun, Sneha Kudugunta, Chang Lan, Katherine Lee, Benjamin Lee, Eric Li, Music Li, Wei Li, YaGuang Li, Jian Li, Hyeontaek Lim, Hanzhao Lin, Zhongtao Liu, Frederick Liu, Marcello Maggioni, Aroma Mahendru, Joshua Maynez, Vedant Misra, Maysam Moussalem, Zachary Nado, John Nham, Eric Ni, Andrew Nystrom, Alicia Parrish, Marie Pellat, Martin Polacek, Alex Polozov, Reiner Pope, Siyuan Qiao, Emily Reif, Bryan Richter, Parker Riley, Alex Castro Ros, Aurko Roy, Brennan Saeta, Rajkumar Samuel, Renee Shelby, Ambrose Slone, Daniel Smilkov, David R. So, Daniel Sohn, Simon Tokumine, Dasha Valter, Vijay Vasudevan, Kiran Vodrahalli, Xuezhi Wang, Pidong Wang, Zirui Wang, Tao Wang, John Wieting, Yuhuai Wu, Kelvin Xu, Yunhan Xu, Linting Xue, Pengcheng Yin, Jiahui Yu, Qiao Zhang, Steven Zheng, Ce Zheng, Weikang Zhou, Denny Zhou, Slav Petrov, and Yonghui Wu. 2023. Palm 2 technical report. *Preprint*, arXiv:2305.10403.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron

McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. 2022. Constitutional ai: Harmlessness from ai feedback. *Preprint*, arXiv:2212.08073.

D. A. Boiko, R. MacKnight, and G. Gomes. 2023. Emergent autonomous scientific research capabilities of large language models. *CoRR*, abs/2304.05332.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *Preprint*, arXiv:2005.14165.

HashiCorp. 2023. Terraform by hashicorp.

Rodrigo Toro Icarte, Richard Valenzano, Toryn Q. Klassen, Phillip Christoffersen, Amir massoud Farahmand, and Sheila A. McIlraith. 2020. The act of remembering: a study in partially observable reinforcement learning. *Preprint*, arXiv:2010.01753.

Anirudh Khatry, Sumit Gulwani, Priyanshu Gupta, Vu Le, Mukul Singh, Ananya Singha, and Gust Verbruggen. 2023. Tstr: Target similarity tuning meets the real world. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10256–10261.

LangChain. 2023. [link].

Michael L. Littman. 1993. An optimization-based categorization of reinforcement learning environments.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *Preprint*, arXiv:2303.17651.

Ken McRae and Michael Jones. 2013. *14 Semantic Memory*, volume 206. Oxford University Press Oxford.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2022. Webgpt: Browser-assisted question-answering with human feedback. *Preprint*, arXiv:2112.09332.

OpenAI. 2023. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Siqi Ouyang and Lei Li. 2023. AutoPlan: Automatic planning of interactive decision-making tasks with large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3114–3128, Singapore. Association for Computational Linguistics.

Fabian Paischer, Thomas Adler, Markus Hofmarcher, and Sepp Hochreiter. 2023. Semantic helm: A human-readable memory for reinforcement learning. In *Neural Information Processing Systems*.

J. S. Park, J. C. O'Brien, C. J. Cai, et al. 2023. Generative agents: Interactive simulacra of human behavior. *CoRR*, abs/2304.03442.

Gabriel Poesia, Alex Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. 2022. Synchromesh: Reliable code generation from pre-trained language models. In *International Conference on Learning Representations*.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7957–7968, Singapore. Association for Computational Linguistics.

C. Qian, X. Cong, C. Yang, et al. 2023. Communicative agents for software development. *CoRR*, abs/2307.07924.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, Bokai Xu, Zhen Zhang, Yining Ye, Bowen Li, Ziwei Tang, Jing Yi, Yuzhang Zhu, Zhenning Dai, Lan Yan, Xin Cong, Yaxi Lu, Weilin Zhao, Yuxiang Huang, Junxi Yan, Xu Han, Xian Sun, Dahai Li, Jason Phang, Cheng Yang, Tongshuang Wu, Heng Ji, Zhiyuan Liu, and Maosong Sun. 2023. Tool learning with foundation models. *Preprint*, arXiv:2304.08354.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Gizem Soğancıoğlu, Hakime Öztürk, and Arzucan Özgür. 2017. Biosses: a semantic sentence similarity estimation system for the biomedical domain. *Bioinformatics*, 33(14):i49–i58.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. 2022. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Tenable. 2023. Terrascan sandbox | tenable.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P Xing, and Zhiting Hu. 2023. Promptagent: Strategic planning with language models enables expert-level prompt optimization. *arXiv preprint arXiv:2310.16427*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. In *Thirty-sixth Conference on Neural Information Processing Systems*.

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.

Lilian Weng. 2023. Llm powered autonomous agents.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework.

Z. Xi, S. Jin, Y. Zhou, et al. 2023. Self-polish: Enhance reasoning in large language models via problem refinement. *CoRR*, abs/2305.14497.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2023. Large language models as optimizers. *Preprint*, arXiv:2309.03409.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. REACT: Synergizing Reasoning and Acting in Language Models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Princeton, NJ, USA. ICLR. *Equal contribution.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023c. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Qinyuan Ye, Maxamed Axmed, Reid Pryzant, and Fereshte Khani. 2023. Prompt engineering a prompt engineer. *Preprint*, arXiv:2311.05661.

Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H. Chi, Quoc V Le, and Denny Zhou. 2023. Take a step back: Evoking reasoning via abstraction in large language models. *Preprint*, arXiv:2310.06117.

D. Zhou, N. Schärli, L. Hou, et al. 2023a. Least-to-most prompting enables complex reasoning in large language models. In *Proceedings of the Eleventh International Conference on Learning Representations (ICLR)*, Kigali, Rwanda. OpenReview.net.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023b. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*.

A Appendix

A.1 METAREFLECTION optimized prompts

In this section, we show some input examples in several tasks for the base model . Specifically our tasks fall into 3 categories: binary classification, multiple choice selection and direct answer matching. As representative examples we take IAC vulnerability detection, casual judgement and HOTPOTQA.

In Figure 4, we illustrate that METAREFLECTION is capable of learning domain-specific instructions. Instructions 2 through 7 specifically address how 'azurerm_container_registry' influences the vulnerability within the code, thereby simplifying the task significantly. Conversely, in Figure 3, a detailed analysis of the prompt generated by PROMPTAGENT reveals an absence of domain-specific information relevant to the Terrascan policy it is intended to evaluate.

```
You are supplied with a terraform model and an Azure defense
policy, both given in dictionary format. The policy is
constructed around the 'Container Registry' resource type.
Thoroughly examine each part of these inputs to understand
and identify their characteristics.

Answer the question: "Does the terraform module comply with
the given policy?" by considering the following instructions
and applying them to your judgment.

Identify the terraform module as 'Compliant' if it meets
either of these two conditions:
1. The resource type specified in the policy is not included
in the terraform module.
2. If the terraform module does contain the resource type,
it must adhere to all the policy regulations.

Conversely, label the terraform module as 'Violating' if it
includes the resource type in question as stated in the
policy but falls short of complying with all the policy
regulations.

Using your understanding of the context and the pattern of
the defense policy dictionary, examine the terraform module
and gauge whether it is 'Violating' or 'Compliant' regarding
the presented policy. Your response should contain only
'Violating' or 'Compliant' with no further elaborations or
explanations.

Reflect on the idea that merely listing the resource type
within the terraform module isn't enough to fulfill
compliance guidelines. It is imperative that to be flagged
as 'Compliant', the module adheres to all policy
regulations. If this is not the case, it should be labeled
'Violating'.

Notably, when the terraform modules do not outline the
resource type specified in the policy, these modules are
automatically determined as 'Compliant'.

Use all the above information to formulate and provide your
evaluation of the terraform model's alignment to the defense
policy.
```

Figure 3: Prompt optimized by PROMPTAGENT

A.2 Vulnerability Detection in IAC Files

Infrastructure-as-Code (IAC) is a popular method of configuring cloud infrastructures, on platforms such as Azure and AWS, using a configuration coding language. These configuration files can declare resources such as virtual machines with specific capabilities, virtual networks and subnets, and data stores. IAC presents an alternative to the traditional ways of configuring cloud infrastructures, such as using a web-based graphical interface. There are numerous IAC platforms currently available for various cloud computing platforms. Here, we focus on Terraform, a leading IAC platform by Hashicorp (HashiCorp, 2023), as well as Azure, the cloud computing platform by Microsoft. Related Terraform resource declarations are grouped together into Terraform modules which act as a basic, reusable configuration component.

```
You are an expert on Terraform and Azure.
You are very good at detecting Terrascan rule violations in
Terraform code.
Answer as concisely as possible.
Identify if the following policy is being violated by the
Terraform code module below. Provide your response as'True'
if you think the policy is violated, 'False' if you think
the policy is not violated.
[Terraform Policy Begin]
.............
[Terraform Policy End]
[Terraform Module Begin]:
```terraform

File Name: acr.tf
Code:
[Terraform Code Start]
resource "random_id" "acr_suffix" {
  byte_length = 8
} ..........
[Terraform Code End]
[Terraform Module End]
Here are additional instructions that were added to avoid
previous mistakes
you made in the past. Follow these instructions as well.
[Instructions Start]

1. Start by identifying the policy details. The policy in
question is "reme_containerRegistryResourceLock" which
pertains to the "azurerm_container_registry" resource type
and requires that locks be implemented for resilience.
2. Analyze the Terraform code to identify the resource types
present. Look specifically for the
"azurerm_container_registry" resource type as this is the
resource type that the policy pertains to.
3. If the "azurerm_container_registry" resource type is not
present in the Terraform code, then the policy cannot be
violated. In this case, you should predict False for policy
violation.
4. If the "azurerm_container_registry" resource type is
present in the Terraform code, then proceed to the next
step.
5. Check if the "azurerm_container_registry" resource has
implemented locks. This can be done by looking for the
"azurerm_management_lock" resource in the Terraform code and
checking if it is associated with the
"azurerm_container_registry" resource.
6. If the "azurerm_container_registry" resource has
implemented locks, then the policy is not violated. In this
case, you should predict False for policy violation.
7. If the "azurerm_container_registry" resource has not
implemented locks, then the policy is violated. In this
case, you should predict True for policy violation.
8. Remember to base your decisions on the evidence present
in the Terraform code and avoid making assumptions about
indirect compliance.
9. Be adaptable and flexible in your approach, allowing for
various scenarios and future updates to the policy or
Terraform code.

[Instructions End]
```

Figure 4: METAREFLECTION inference prompt for IAC

Task. Cloud infrastructures are prone to security vulnerabilities such as open ports and exposed administrator accounts (Tenable, 2023). Vulnerability detection via static analysis of IAC files is a hard problem due to the expressivity of the configuration
```

Description: *Ensure that Azure Virtual Network subnet is configured with a Network Security Group*

Definition:

```
{{.prefix}}noSecurityGroupAssociated[retVal] {
  vn := input.azurerm_virtual_network[_]
  vn.type = "azurerm_virtual_network"
  object.get(vn.config, "subnet", \
     "undefined") != "undefined"
  not sgExists(vn.config)

  traverse = "subnet[0].security_group"
  retVal := {
    "Id": vn.id,
    ...
    "Attribute": "subnet.security_group",
    "Expected": "${<security_group_name>.id}",
    "Actual": ""
  }
}

sgExists(cfg) { ... }
<56 lines altogether>
```

Figure 5: Rego code for the Terrascan policy `reme_noSecurityGroupAssoc`.

language, the complexity of configurations and the diversity of the resources being handled across multiple infrastructure providers (e.g., Amazon AWS and Microsoft Azure). Further, Terraform uses a low-resource language - HashiCorp Configuration Language (HCL).

Task. Terrascan (Tenable, 2023) is a static analyzer for detecting security vulnerabilities in Terraform modules, and supports over 500 security policies, including 178 policies specific to Azure. Figure 5 shows the description and definition of a Terrascan policy that checks if every Azure virtual network subnet is configured with a corresponding network security policy. Note that the Terrascan policy is *syntactic*, i.e., it is checking for a declaration of an `azurerm_virtual_network` with a field named `subnet`, and so on. Hence, Terrascan-like static analysis based vulnerability detection is fragile and prone to both false positives and false negatives due to being sensitive to syntax. The task at hand is to check if a given Terraform module violates a given Terrascan policy.

Data collection. We collected 202 Terraform modules by mining GitHub repositories for IAC code written in HCL. These repositories corresponded to a diverse range of applications including load balancers, machine learning operations managers, and domain-specific data-stores. For policies, we selected the 10 most commonly violated Terrascan policies. Of the 2020 module-policy pairs, we eliminated a significant fraction of cases where the policies were not applicable to the module. For example, if the policy was for a specific resource type and the module did not contain declarations of that resource type, the pair was eliminated. After this process, we were left with 648 module-policy pairs, for which we manually annotated whether the module violated the policy (see Table 3 for the exact breakdown). Note that this ground-truth annotation was with respect to the description of Terrascan policy, not the definition—that is, we use the intention behind the policy, not the letter of the definition. That is, we do not take the output of Terrascan as ground truth as it can be inaccurate, and instead manually examine if the policy (as per description) is violated. This data was then split into train and test sets in a $40:60$ ratio per policy, taking care to balance the vulnerable and non-vulnerable classes.

ProTeGi optimized prompt

When answering, examine the question and context provided to offer an exact and accurate response without adding irrelevant details or making assumptions. Use the contextual details to answer specifically and pay attention to the language and form of the question. If a name is required in the answer, ensure to reproduce the exact name as it appears in the context without making alterations. Numeric or yes/no questions should be answered in a clear and straight-to-the-point manner, with no need for punctuation or capital letters. When the answer requires a term or phrase, select the most relevant and specific one as it appears in the context. Always maintain the form (singular or plural) used in the question when formulating answer. Answer must be based solely on the context and question provided, avoid straying from this by adding extra information unless it is specifically requested in the question.

PromptAgent optimized prompt

Your task is to provide an accurate answer to a question based on the given context. Be sure to consider all relevant information, not just those closest or directly related to the specifics of the question. Your task involves parsing both the local and larger context, and forming your response appropriately. The required answer may be indirectly linked to the focus of the question as justified by the overall context. It is important to process the complex or abstract links between the answer and the question within the context. Respond precisely, preferably with a single word or short phrase, but avoid forming full sentences or supplying extra information that doesn't serve the question. Remember, relevance is vital—your answer should reflect the content, essence, and nuance of the question while using the context as your guide.

MetaReflection learned instructions:

Be mindful of the exact phrasing in the question and the context. Even if your answer is technically correct, it may be marked as incorrect if it does not match the exact phrasing used in the context or the expected answer. Avoid adding extra information that is not directly asked in the question. Pay attention to singular and plural forms in both the question and the context. Be aware of implicit information in the context and make logical inferences when necessary. This will help you provide more accurate answers. When answering questions about historical figures or events, use the exact title or name mentioned in the context. When the question asks for numerical data, provide the number alone without additional context or explanation. When a question asks for a common characteristic or profession of multiple individuals, provide the answer in singular form even if it applies to multiple individuals. If the question asks for a specific individual or entity, provide only that individual or entity in your answer, even if the context mentions a connection to another individual or entity. Avoid including additional individuals or entities unless explicitly asked for in the question. When the question asks for a concept, provide the concept in the language used in the context, unless the question specifically asks for a translation or explanation. When answering questions about international competitions or events, include all relevant details from the context in your answer, especially when it comes to the country represented by the individual. When the context provides multiple names for the same entity, use the most detailed and specific name in your answer

Figure 6: Snapshot of the optimized prompt learnt by the baselines and METAREFLECTION on HOTPOTQA task

Experimental setup. As a baseline language agent, we use GPT-4 with an appropriate prompt that provides the code of the Terraform module and the description of the Terrascan policy, and asks if the module is vulnerable. While training, the agent is given a 0-1 feedback on whether its response is correct or not, and the model is asked to self-reflect if the response is incorrect. For each policy, we run the METAREFLECTION algorithm on the training

```
// Method 1: For inline defintions
resource "azurerm_virtual_network" "example" {
    ...
    subnet {
        ...
        security_group = ...
    }
}

// Method 2: Explicitly declared association
resource "azurerm_subnet_nsg_association" {
   subnet_id = ...
   network_security_group_id = ...
}
```

(a) Associating a subnet with a NSG

```
resource "azurerm_virtual_network" "vnet" { ... }
resource "azurerm_subnet" "subnet" {
    name = "subnet1" ...
}
resource "azurerm_network_interface" "nic" {
    network_security_group_id = ...
    ip_configuration { subnet_id = "subnet1" }
}
resource "azurerm_virtual_machine" "..." {
    network_interface_ids = [ "nic" ]
    ...
}
```

(b) NSG associated with a VM's network interface instead of subnet.

```
   4.       Remember    that    the    association
between "azurerm_virtual_network" and a NSG
may not be direct.     It could be done
through a separate resource block such as
"azurerm_subnet_nsg_association"
...
7. Do not confuse NSG associations with network
interfaces of VMs and the subnet of the Azure
Virtual Network. The policy specifically requires
the NSG be associated with the subnet.
```

(c) Instructions learned through meta-reflection

Figure 8: `reme_noSecurityGroupAssociated`: Checking Subnet-NSG associations

```
You are an expert in [Task]. Given the following task description [and
examples] come up with a set of instructions that can help you perform the
task effectively.

Task Description: ...
```

Figure 7: Prompt for generating task-specific instructions in the LLMINSTRUCTION baseline

set and report the accuracy numbers for both the baseline agent and the agent with the instructions learned through METAREFLECTION. We also compare to LLMINSTRUCTION as another baseline—here the language model is asked to come up with instructions for a task given its description (Figure 7), and then these instructions are provided when the task is being performed.

Results. The results of the experiment are summarized in Tables 3 and 6 (last 2 cols). On the whole, across all policies, meta-reflection shows a $12-17\%$ accuracy improvement over the baselines depending on the batch size. As Table 3 shows, meta-reflection provides consistent gains in accuracy for all policies over the GPT-4 baseline, with 32% in the best case. The precision with METAREFLECTION is significantly better for all policies, while the recall decreases for some.

Exemplar case. We discuss the case of security policy `reme_noSecurityGroupAssociated` from Figure 5, i.e., that all Azure virtual network subnets are configured with a network security group (NSG). The main difficulty here is that HCL and Terraform offer many different ways of (a) associating a subnet with a virtual network, and (b) associating a NSG with a subnet. By default, the baseline GPT-4 agent fails to handle certain ways of specifying these associations, while spuriously assuming certain other associations. In Figure 8a, the baseline consistently failed to recognize a subnet-NSG association expressed using Method 2, i.e., using an explicitly declared association. On the other hand, it mis-identified declarations similar to the one in Figure 8b as valid subnet-NSG associations—here, the NSG is associated with a virtual machine's network interface (that is connected to the subnet) instead of the subnet itself. These limitations lead to both false positives and false negatives. With meta-reflection, we are able to learn the instructions in Figure 8c, using which the agent easily handles these kinds of cases.

Discussion. As the above exemplar case shows, METAREFLECTION is able to learn very domain-specific instructions to fix both false positives and false negatives. Other instructions include aspects like handling of wildcards for port numbers, step-by-step strategies for specific policies, etc. Note that these instructions not only include planning (or trajectory directing) instructions, but also grounding instructions—i.e., external facts that are not initially available.

In general, the experimental results show that meta-reflection is able to reduce the number of errors, i.e., improve the accuracy across a broad range of cases. However, one noticeable issue from the above results is the drop in recall for several policies. While the high recall in the baseline is artificial, coming at the cost of low precision, this is still an important issue to address. Our 0-1 feedback to the self-reflection agent does not state that

	HotpotQA						IaC vulnerability detection	
	CoT (GT)		CoT (Distractor)		ReAct			
	Acc.	σ	Acc.	σ	Acc.	σ	Acc.	σ
GPT-4	20.67	3.79	32.67	3.21	19.58	1.91	73.36	0.012
LLMInstruction	20.67	4.51	30.67	9.81	27.5	2.5	73.31	0.43
MetaReflection (batch size = 1)	24.67	3.51	38.00	2.65	30.41	0.72	87.17	0.63
MetaReflection (batch size = 2)	50.33	0.58	48.67	2.52	**35.00**	1.25	85.09	0.4
MetaReflection (batch size = 4)	**52.00**	2.00	**50.67**	1.53	31.67	1.44	**90.18**	0.28

Table 6: Results on HotpotQA and IaC with GPT-4

false negatives are worse than false positives in the security domain. In the future, we plan to explore domain-specific feedback and self-reflection mechanisms that can account for the nature of errors, as well as better versions of the ShowsImprovement function that are aware of such domain-specific preferences.

A.3 HotpotQA

HotpotQA (Yang et al., 2018) is an open-domain factual question answering dataset consisting of 113K question-answer pairs. The original paper proposes to use the data in 2 settings: (a) *Distractor* setting - where each question is to be answered using 10 wikipedia article excerpts; and (b) *Full-Wiki* setting which is a retrieval and reasoning task, where a given question is supposed to be answered after retrieving relevant context from wikipedia. Notably, an answer is marked correct only if it matches *exactly* with the ground truth.

Similar to Shinn et al. (Shinn et al., 2023), we design the following agents that operate over the dataset: (a) ReAct- for the *Full-Wiki* setting (b) CoT (Distractor) - for the *Distractor Distractor* setting (c) CoT (GT) - a variant of CoT (Distractor) with access to only ground truth articles.

Data Sampling. For each agent setting, we adversarialy sample subsets of the HotpotQA train split of 90K samples to create train and test sets. To perform adversarial sampling, we first identify samples where the base agent fails consistently in a given setting. On these failing examples we perform upto 3 self-reflection trials to get the model to the right response. If the agent is not able to get to the correct response even after self-reflection, we discard these samples. This strategy ensures that we get a set of hard examples in which the agents fail most of the times to get to the right answer in a single try, while also making sure that we filter examples that may be noisy due to missing context, incorrect questions etc. To account for randomness and given our computational budget, we sample 40 and 80 examples for the ReAct train set and test set respectively. For CoT settings, we pick 50 and 80 example respectively.

Experimental setup. We reuse the CoT agent from (Wei et al., 2023) for the chain-of-thought experiments and use a re-implementation of (Yao et al., 2023c) for the ReAct experiments. The ReAct agent is allowed at most 6 Action steps after which the trajectory is automatically determined to be a failure. Similar to Section A.2, we evaluate HotpotQA configurations for: (a) MetaReflection with batch sizes 1, 2, and 4; and (b) GPT-4 and LLMInstruction as baselines. In addition to this, we also evaluate variants of the agents powered by GPT-3.5-Turbo instead of GPT-4, while using GPT-4 for MetaReflection.[1]

Results. We find that the generated MetaReflection instruction consistently improved performance across different agent settings for HotpotQA. In Table 6, we present results using GPT-4 for both the agents and MetaReflection. We observe that MetaReflection help us achieve gains up to 32% for CoT (GT), 18% for CoT (Distractor), and 15.5% for ReAct, over the respective test sets. Interestingly, higher batch sizes almost always help, reinforcing the importance of batching as observed in related works (Ouyang and Li, 2023; Ye et al., 2023).

In Table 7, we report results when using GPT-3.5-Turbo to power the client agents. We see gains of upto 21% gains for CoT (GT), 12% for CoT (Distractor) and 18.33% for the ReAct case. Here, we observe that batching doesn't strictly improve the performance. Examining the data qualitatively, this difference can be attributed to the nature of the instructions generated in the two settings. In general we observe that with a small batch size,

[1] A similar experiment on the IaC domain wasn't possible due to large context length of the Terraform modules

Inference without meta-reflection

Solve a question answering task by having a Thought, then Finish with your answer. Thought can reason about the current situation. Finish[...] returns the answer and finishes the task.
Question: What is one name for the Cassava after it is marketed as a paste and often served with palm nut soup?

Thought: The question is asking for a name of cassava when it is marketed as a paste and often served with palm nut soup. From the context provided, I can see that cassava paste is referred to as "Akpu" in Nigeria and is often eaten with soups such as Egusi soup. Although palm nut soup is not explicitly mentioned in the context of Akpu, it is a type of soup that could be served with cassava paste, as indicated by the information about palm nut soup being used to accompany dishes like fufu, which is similar to Akpu in that they are both made from cassava.
Action : Finish[Akpu]

Inference with meta-reflection

Answer a question interleaving Thought, Action, and Observation steps. Actions can be Search[...], Lookup[...], ...

Instructions: ...
If the context suggests multiple valid answers, choose the one that best matches the question's wording and the most direct information provided.
In cases of ambiguity or multiple interpretations, provide an answer that adheres to the most common understanding of the question's terms and context.
...
Question: What is one name for the Cassava after it is marketed as a paste and often served with palm nut soup?

...
Thought: The question is asking for a name of cassava when it is marketed as a paste and often served with palm nut soup. From the context provided, I can see that cassava paste is referred to as "Akpu" in Nigeria and is often eaten with Egusi soup. However, since the question specifically mentions palm nut soup, I should look for a name that is associated with cassava paste served with palm nut soup. The context also mentions "Plakali" as a staple food made of cassava dough and eaten with palm nut soup in Ghana. However, "Plakali" is described as cassava dough cooked in hot water, not as a marketed paste. The most direct match for cassava marketed as a paste and associated with palm nut soup is "Fufu," which is mentioned as a staple food often served with palm nut soup.
Action: Finish[Fufu]

Figure 10: METAREFLECTION in the HOTPOTQA domain with COT style of reasoning

METAREFLECTION produces a large amount of very specific instructions. On the contrary batching helps generalize these instructions into more widely applicable rules. GPT-4, being more powerful than GPT-3.5-TURBO, is able to better follow these abstract instructions, while specific instructions work better for GPT-3.5-TURBO.

```
// Chain-of-thought

(A) Provide direct and concise responses to the question, using precise language that matches the specificity and terminology of the question, including singular or plural forms and definite articles as needed.

(B) If the context suggests multiple valid answers, choose the one that best matches the question's wording and the most direct information provided.

// React

(C) When a question asks for a specific detail such as a 'full name', ensure to find and provide that exact information. Don't make assumptions based on limited or incomplete information.

(D) If you're not finding the desired information or stuck in a loop of looking up a keyword, consider changing the keyword and search strategy. The information might be located further down the page.

(E) When a question involves comparison, such as 'who received more rewards', ensure to search for each entity individually, gather all necessary information, and then make a comparison based on the data found.

(F) Be mindful of potential spelling errors or variations in the names of entities. If a search for a specific term doesn't yield results, consider possible alternative spellings or forms of the term.
```

Figure 9: Instructions learned via the METAREFLECTION technique for the COT and REACT reasoning on HOTPOTQA.

Exemplar case. Consider an example question from Figure 10. The question seeks information about the product made from Cassava and served with palm nut soup. The context presented within the COT (Distractor) setting includes articles about *Akpu* and *Fufu*, both of which are quite similar, being made from Cassava paste. However, the key distinction lies in *Fufu* being served with palm nut soup, while *Akpu* is served with Esupi soup. The baseline COT agent returns the incorrect response on this question: it is distracted by the related but incorrect articles, and makes an incorrect assumption and jumps to the wrong conclusion. The METAREFLECTION technique learns an instruction that suggests looking for multiple valid answers and selecting the one most related to the question. When inferring with the meta-reflection instructions, it is clear from the thought that the agent did encounter the misleading answers, but was able to produce the right one by focusing on the specific key point "served with palm nut soup" mentioned in the question.

Similarly, in the REACT case (see Figure 1), we see the learned instruction enhancing search strategy by looking into the information further down the page rather looping around. This rule further aids the model in successfully concluding the trial where it was previously failing. The model uses the rule to explicitly guide the action space to look further down the context page and look up the right keyword, leading to the correct response, Bruce Dickinson. In contrast, in the baseline attempt, it ran out of trials by getting stuck in a loop.

Discussion. As we can see from the results, meta-reflection can produce significant improvements in accuracy in the question answer setting. This is especially promising given that the dataset was sampled using an adversarial sampling technique. The HOTPOTQA domain also shows the diversity of instructions learned by METAREFLECTION—a small selection of instructions learned in the COT and REACT settings are shown in Figure 9 We have instructions that are: i. specifically tuned to satisfy the overly strict rubric of the HOTPOTQA dataset (A); ii. domain-specific instructions for specific one-step actions in a RL trajectory (C); iii. directly the high-level strategy to be taken by the trajectory (D, E); and iv. for disambiguating answers (B) and questions (E). Further, the results on GPT-3.5-TURBO experiments reveal that METAREFLECTION can be useful to enhance the performance

	CoT (GT)		CoT (Distractor)		ReAct	
	Acc.	σ	Acc.	σ	Acc.	σ
GPT-3.5-Turbo	23.00	2.65	30.00	5.00	8.33	2.60
LLMInstruction	25.67	6.03	31.00	10.00	20.83	4.83
MetaReflection (batch size = 1)	29.0	1.00	**41.67**	2.52	**26.67**	4.73
MetaReflection (batch size = 2)	38.67	2.52	30.67	1.15	17.08	2.60
MetaReflection (batch size = 4)	**44.33**	0.58	39.33	1.15	22.08	5.64

Table 7: Results on HotpotQA with GPT-3.5-Turbo powering the agents and GPT-4 for reflections and MetaReflection

of smaller models by providing instructions rich in specific insights from a more powerful LLMs like GPT-4. This shows some resemblance to task-specific distillation and can be interesting to explore further in future works.