META-PROMPT OPTIMIZATION FOR LLM-BASED SEQUENTIAL DECISION MAKING

Mingze Kong[1], **Zhiyong Wang**[2], **Yao Shu**[3], **Zhongxiang Dai**[1]*
[1]The Chinese University of Hong Kong, Shenzhen, [2]The Chinese University of Hong Kong,
[3]The Hong Kong University of Science and Technology (Guangzhou)
mingzekong@cuhk.edu.cn, zhiyongwangwzy@gmail,
shuyao95@gmail.com, daizhongxiang@cuhk.edu.cn

ABSTRACT

Large language models (LLMs) have recently been employed as agents to solve sequential decision-making tasks such as Bayesian optimization and multi-armed bandits (MAB). These works usually adopt an LLM for sequential action selection by providing it with a fixed, manually designed *meta-prompt*. However, numerous previous works have found that the prompt has a significant impact on the performance of the LLM, which calls for a method to automatically optimize the meta-prompt for LLM-based agents. Unfortunately, the non-stationarity in the reward observations during LLM-based sequential decision-making makes meta-prompt optimization highly challenging. To address this challenge, we draw inspirations from *adversarial bandit* algorithms, which are inherently capable of handling non-stationary reward observations. Building on this foundation, we propose our *EXPonential-weight algorithm for prompt Optimization* (`EXPO`) to automatically optimize the task description and meta-instruction in the meta-prompt for LLM-based agents. We also extend `EXPO` to additionally optimize the exemplars (i.e., history of interactions) in the meta-prompt to further enhance the performance, hence introducing our `EXPO-ES` algorithm. We use extensive experiments to show that our algorithms significantly improve the performance of LLM-based sequential decision-making.

1 INTRODUCTION

The strong capabilities of LLMs have spurred significant recent interests in adopting them as agents to solve sequential decision-making problems, such as multi-armed bandits (MAB) (Krishnamurthy et al., 2024), Bayesian optimization (BO) (Yang et al., 2024) and reinforcement learning (RL) (Dai et al., 2024). Specifically, these methods often use an LLM to sequentially select the actions by providing it with a specially designed prompt, which we refer to as the *meta-prompt*. The meta-prompt often contains several components, such as the *task description*, the *meta-instruction* (which is used to instruct the LLM to select an action in every step), the *history of interactions* with the environment, among others. The previous methods have all adopted a fixed, manually designed meta-prompt for the LLM-based agent throughout the entire sequential decision-making process. However, numerous previous works have highlighted that the output text generated by LLMs is heavily dependent on its input prompt (Zhou et al., 2023). Therefore, using fixed, manually designed meta-prompt may significantly limit the performance of the LLM-based agents, because handcrafted prompts are often far from optimal (Lin et al., 2024b). This naturally begs the question: *can we automatically optimize the meta-prompt for LLM-based agents to enhance their performance?*

The sensitivity of LLM-generated text to its input prompt has given rise to many recent works on *automated prompt optimization*, among which a representative line of works have adopted the method of multi-armed bandits (MAB) to automatically optimize the prompt (Lin et al., 2024b; Wu et al., 2024; Lin et al., 2024a). Unfortunately, the problem of meta-prompt optimization for LLM-based agents presents significant challenges compared to traditional prompt optimization. This is mostly due to the *non-stationarity in the observed rewards* during the LLM-based sequential decision-making

*Corresponding author.

process. Specifically, as the LLM-based agent engages in more interactions with the environment, its state in the environment changes, making its observed rewards non-stationary. For example, in MAB (Krishnamurthy et al., 2024) and BO (Yang et al., 2024), the observed rewards in later iterations (i.e., after the agent has accumulated significant experience in the environment) tend to be higher than those obtained in initial iterations. Similarly, in RL (Dai et al., 2024), rewards are typically dependent on both the state and action. However, since the state of the LLM-based agent evolves across iterations, this also results in non-stationarity in the observed rewards. As a consequence of the non-stationarity, for the same meta-prompt (e.g., the same task description and meta-instruction), its corresponding observed reward is highly likely to be dynamically changing across different iterations. This is in stark contrast to classical prompt optimization, in which the reward or score for a prompt remains stationary across iterations. As a result, this renders the previous works on prompt optimization (such as those based on MAB (Lin et al., 2024b; Wu et al., 2024)) inapplicable, and hence calls for novel algorithmic designs to solve the problem of meta-prompt optimization for LLM-based agents. To this end, we draw inspirations from the field of *adversarial bandits* (Lattimore & Szepesvári, 2020).

In adversarial bandits, for each arm, the reward observations when the arm is pulled are chosen by an adversary, i.e., they are allowed to change in an arbitrary way across different iterations. Therefore, the reward observations can be significantly non-stationary. This is considerably different from classical stochastic MAB, in which the reward observations for an arm are sampled from a fixed stationary distribution. Therefore, the ability of adversarial bandits to handle non-stationary reward observations makes it an ideal candidate for meta-prompt optimization for LLM-based agents. Specifically, drawing inspirations from the EXP3 algorithm for adversarial bandits, we introduce our *EXPonential-weight algorithm for prompt Optimization* (EXPO) to optimize the task description and meta-instruction in the meta-prompt of an LLM-based agent.[1]

In addition to the task description and meta-instruction, the *history of interactions* with the environment (which we also refer to as the *exemplars*) is also a crucial component in the meta-prompt which exerts a considerable impact on the performance of LLM-based agents. Existing works often adopt simple heuristic approaches to decide how to incorporate the exemplars into the meta-prompt, including which subset of exemplars is included and their ordering in the meta-prompt. Previous works on in-context learning (ICL) have found that in addition to their contents, the ordering of the exemplars also has a significant impact on the performance of LLMs (Lu et al., 2022). Therefore, in addition to optimizing the task description and meta-instruction, we also extend our EXPO algorithm to additionally optimize both the subset of exemplars included in the meta-prompt and their ordering. However, the optimization of the task description and meta-instruction in every iteration in our EXPO makes the optimization of exemplars non-stationary as well. Specifically, for the same subset of exemplars with a fixed ordering, their reward observations are usually non-stationary, because the task description and meta-instruction selected by our EXPO algorithm are highly likely to vary across different iterations. To this end, we extend our EXPO algorithm to additionally use a separate adversarial bandit method to optimize the exemplars (i.e., the interaction history) in the meta-prompt for LLM-based agents, and hence introduce our *EXPO with Exemplar Selection* (EXPO-ES) algorithm.

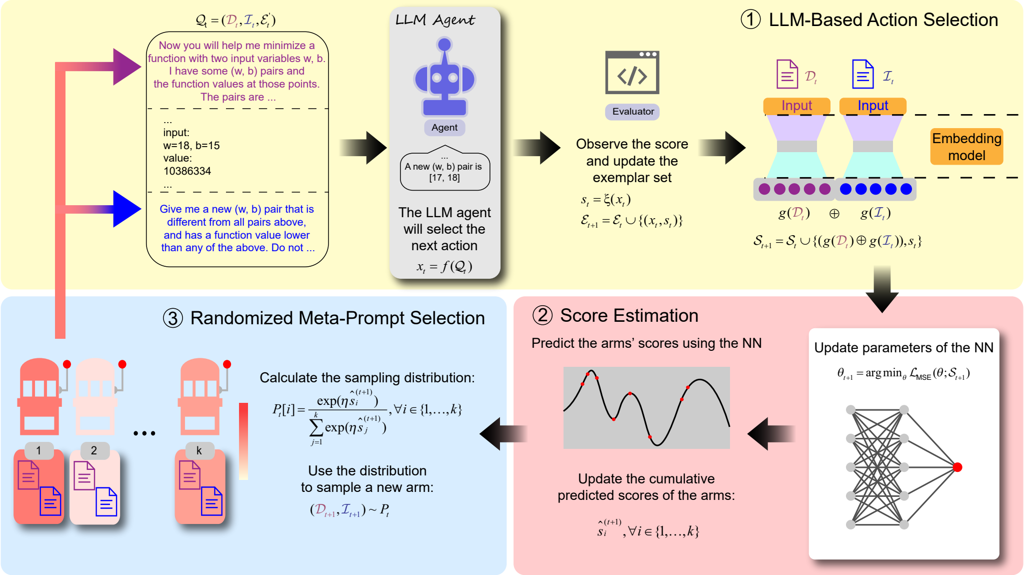
We use extensive experiments to show that our EXPO algorithm significantly improves the performance of the LLM-based BO algorithm from Yang et al. (2024) (Sec. 4.1) and the LLM-based MAB algorithm from Krishnamurthy et al. (2024) (Sec. 4.2). Furthermore, in tasks where the exemplars provide crucial information for the LLM-based agent, our EXPO-ES algorithm further enhances the performance of EXPO via automated exemplar selection (Sec. 4.1). We also perform ablation study to unveil other interesting insights about our algorithms in Sec. 5.

2 Problem Setting

We use *arms* to represent meta-prompts, and use *actions* to denote the actions selected by an LLM-based agent.

Consider an algorithm which uses an LLM to perform a sequential decision-making task by sequentially instructing the LLM to select an action in every iteration. A representative example of such algorithms is the *Optimization by PROmpting* (OPRO) algorithm from Yang et al. (2024). OPRO

[1] Note that although here we only consider optimizing the task description and meta-instruction, the other components contained in the meta-prompt (e.g., some information from previously completed related tasks) can also be optimized in a similar fashion.



$\mathcal{Q}_t = (\mathcal{D}_t, \mathcal{I}_t, \mathcal{E}'_t)$

Now you will help me minimize a function with two input variables w, b. I have some (w, b) pairs and the function values at those points. The pairs are ...

...
input:
w=18, b=15
value:
10386334
...

Give me a new (w, b) pair that is different from all pairs above, and has a function value lower than any of the above. Do not ...

LLM Agent
Agent
... A new (w, b) pair is [17, 18]
The LLM agent will select the next action
$x_t = f(\mathcal{Q}_t)$
Evaluator
Observe the score and update the exemplar set
$s_t = \xi(x_t)$
$\mathcal{E}_{t+1} = \mathcal{E}_t \cup \{(x_t, s_t)\}$
① LLM-Based Action Selection
$\mathcal{D}_t$
$\mathcal{I}_t$
Input
Input
Embedding model
$g(\mathcal{D}_t)$ ⊕ $g(\mathcal{I}_t)$
$\mathcal{S}_{t+1} = \mathcal{S}_t \cup \{(g(\mathcal{D}_t) \oplus g(\mathcal{I}_t)), s_t\}$
② Score Estimation
Predict the arms' scores using the NN
Update parameters of the NN
$\theta_{t+1} = \arg\min_\theta \mathcal{L}_{\text{MSE}}(\theta; \mathcal{S}_{t+1})$
Update the cumulative predicted scores of the arms:
$\hat{s}_i^{(t+1)}, \forall i \in \{1, \ldots, k\}$
③ Randomized Meta-Prompt Selection
1
2
...
k
Calculate the sampling distribution:
$P_t[i] = \frac{\exp(\eta \hat{s}_i^{(t+1)})}{\sum_{j=1}^{k} \exp(\eta \hat{s}_j^{(t+1)})}, \forall i \in \{1, \ldots, k\}$
Use the distribution to sample a new arm:
$(\mathcal{D}_{t+1}, \mathcal{I}_{t+1}) \sim P_t$

Figure 1: Illustration of our EXPO algorithm. We use purple to denote the task description and blue to represent the meta-instruction.

aims to solve an optimization problem, i.e., to find $x^* = \arg\min_x f(x)$. To achieve this, in every iteration t, OPRO uses an LLM to select a batch of B input queries $\{x_{t,1}, \ldots, x_{t,B}\}$, after which their corresponding scores $\{s_{t,1}, \ldots, s_{t,B}\}$ are observed. When instructing the LLM to select the input queries, the meta-prompt $\mathcal{Q}$ given to the LLM contains a number of important components, including *a fixed task description* $\mathcal{D}$, *a fixed meta-instruction* $\mathcal{I}$, and *a sequence of exemplars* $\mathcal{E}_t$ corresponding to a subset of the observations (i.e., pairs of input queries and observed scores) collected so far. The same paradigm of LLM-based sequential decision-making has also been adopted by other works, such as the LLM-based MAB algorithm from Krishnamurthy et al. (2024) (more details in Sec. 4.2).

In this work, our first algorithm, EXPO (Sec. 3.1), dynamically optimize the task description $\mathcal{D}$ and meta-instruction $\mathcal{I}$ (i.e., selects a new $\mathcal{D}_t$ and $\mathcal{I}_t$ in every iteration t), in order to improve the efficiency and effectiveness of optimization. We also extend our EXPO to derive the EXPO-ES algorithm (Sec. 3.2), which additionally optimizes the sequence of exemplars $\mathcal{E}_t$ to further improve the optimization performance. We use $g(\cdot)$ to denote a pre-trained embedding function, which maps some input text to its corresponding continuous representation. We separately obtain the embeddings of the task description $g(\mathcal{D}_t)$, the meta-instruction $g(\mathcal{I}_t)$ and the exemplar sequence $g(\mathcal{E}_t)$. Based on the embeddings, in every iteration, we use the current history of selected meta-prompts and their scores to train a neural network (NN), which can then be used to predict the scores of every meta-prompts in the domain. We denote this NN as $\mathcal{M}(g(\cdot); \theta)$, in which θ represents the NN parameters.

Adversarial Bandits. In adversarial bandits, the goal is to compete against the best arm *in hindsight* (Lattimore & Szepesvári, 2020). Consider an MAB problem with k arms (i.e., meta-prompts). For each arm $i = 1, \ldots, k$, denote its corresponding sequence of rewards (i.e., scores) in T iterations as $\{r_{t,i}\}_{t=1,\ldots,T}$. The best arm in hindsight is then defined as $i^* = \arg\max_{i=1,\ldots,k} \sum_{t=1}^{T} r_{t,i}$. Then, the goal of an adversarial bandit algorithm (which selects arm A_t in iteration t) is to minimize the following definition of regret: $R_T = \sum_{t=1}^{T} r_{t,i^*} - \sum_{t=1}^{T} r_{t,A_t}$.

Adversarial Bandits for LLM-Based Agents. LLM-based sequential decision-making methods often aim to maximize either *(a)* the cumulative rewards (e.g., the LLM-based MAB algorithm from Krishnamurthy et al. (2024)) or *(b)* the final reward (e.g., OPRO from Yang et al. (2024)). In the former case of cumulative reward maximization, the overall rewards/scores for the best arm i^* are higher than the other arms. In the latter case, we implicitly assume that the arm with the largest final reward after T iterations also has large rewards across all iterations in general. As a result, in both cases, *the observed rewards of an arm (i.e., the observed scores of a meta-prompt) in every iteration are indicative of the quality of the arm (i.e., the meta-prompt)*. So, when training the NN $\mathcal{M}(g(\cdot); \theta)$ (for score prediction) using the history of the selected meta-prompts and their observed scores, we simply use the scores (i.e., rewards) as the labels in the training set. This simple design helps our algorithms achieve strong performance in our experiments (Sec. 4).

3 ALGORITHMS

3.1 THE EXPO ALGORITHM (ALGO. 1)

Our EXPO is used to dynamically optimize the task description $\mathcal{D}$ and the meta-instruction $\mathcal{I}$.

Domain Generation. At the beginning of our algorithm, we start by generating the domain of task descriptions and meta-instructions. Following the previous works on prompt optimization (Zhou et al., 2023; Lin et al., 2024a;b), we use an LLM to rephrase an initial task description $\mathcal{D}_0$ (resp. initial meta-instruction $\mathcal{I}_0$) to generate a domain of k_1 task descriptions (resp. k_2 meta-instructions). This results in a domain size of $k = k_1 \times k_2$. We defer more details on domain generation to App. B.1. We treat the combination of a task description $\mathcal{D}$ and a meta-instruction $\mathcal{I}$ in the domain as an *arm*, i.e., our adversarial bandit problem has k arms. In addition to jointly optimizing $\mathcal{D}$ and $\mathcal{I}$, we have also evaluated the performance of optimizing them separately. The results show that jointly optimizing these two components leads to better performance.

① **LLM-Based Action Selection (lines 3-7 of Algo. 1).** At the beginning of every iteration t, we firstly use the current task description $\mathcal{D}_t$, meta-instruction $\mathcal{I}_t$ and exemplar sequence $\mathcal{E}'_t$ selected at the end of the last iteration $t-1$ (more details below) to construct a meta-prompt $\mathcal{Q}_t = (\mathcal{D}_t, \mathcal{I}_t, \mathcal{E}'_t)$ (line 3). Then, we use $\mathcal{Q}_t$ as the input prompt to the LLM $f(\cdot)$ to select the next action x_t and collect its score s_t (lines 4-5). After that, we update the set of exemplars $\mathcal{E}_t$ and the meta-prompt-score set $\mathcal{S}_t$ (lines 6-7).

② **Score Estimation (lines 8-9).** In the classical EXP3 algorithm for adversarial bandits with a finite number of arms, the cumulative sum of *the observed rewards* of every arm is used to construct the arm sampling distribution through an exponential-weight mechanism (Lattimore & Szepesvári, 2020). However, in problems where the number of arms is excessively large (e.g., our problem of meta-prompt optimization), the reward observations for many arms are not available. Therefore, the cumulative sum of *the estimated rewards* of every arm is often used instead to construct the sampling distribution (Lattimore & Szepesvári, 2020). Therefore, we firstly estimate the scores of all k arms (i.e., meta-prompts) in the domain and then use these score estimates to derive an arm sampling distribution for our EXPO. A number of recent works have shown that using a neural network (NN) (which takes the pre-trained embedding $g(\cdot)$ as input) for score/reward estimation leads to powerful prompt optimization algorithms (Lin et al., 2024a;b; Wu et al., 2024). Therefore, we also adopt an NN $\mathcal{M}(g(\cdot);\theta)$ for score estimation in our EXPO. Specifically, in every iteration t, we use the history of selected meta-prompts and their scores, denoted as $\mathcal{S}_{t+1}$ (line 7 of Algo. 1), to train an NN by minimizing the mean-squared error (MSE) loss (line 8 of Algo. 1). The trained NN with parameters θ_{t+1} can then be used to estimate the score of every arm (i.e., every combination of task description and meta-instruction) in the domain. For every arm, its estimated score is then added to its corresponding *cumulative sum of score estimates* $\hat{s}_i^{(t+1)}$ (line 9 of Algo. 1). Note that every term in the cumulative sum $\hat{s}_i^{(t+1)}$ represents our score estimate for arm i *in a particular iteration* t, i.e., our estimated score for arm i from an NN trained using the observation history *up to iteration* t. The updated cumulative sums of score estimates $\hat{s}_i^{(t+1)}$ for all k arms are then used for randomized arm (i.e., meta-prompt) selection, which we discuss next.

③ **Randomized Meta-Prompt Selection (lines 10-12).** After the cumulative sum $\hat{s}_i^{(t+1)}$ of every arm i is updated, we follow the EXP3 algorithm (Lattimore & Szepesvári, 2020) and use the cumulative sums to construct a distribution following Equation equation 2. Then, we use this distribution to randomly sample the next arm, i.e., the next task description $\mathcal{D}_{t+1}$ and meta-instruction $\mathcal{I}_{t+1}$ (line 11 of Algo. 1). *Randomization* is a key principle in adversarial bandits (Lattimore & Szepesvári, 2020), and the randomization involved in our arm selection strategy is crucial for the ability of our EXPO to deal with non-stationary reward observations. The heuristic to select a sequence of exemplars $\mathcal{E}'_{t+1}$ (line 12 of Algo. 1) is often specified by the LLM-based sequential decision-making algorithm (Yang et al., 2024). We discuss more details on this, as well as the extension of our EXPO algorithm to automatically select $\mathcal{E}'_{t+1}$, in Sec. 3.2.

Algorithm 1 EXPO

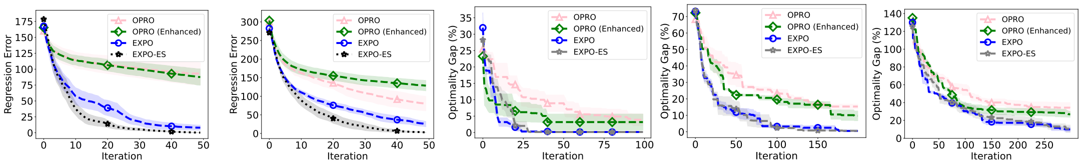
```
input : Initial task description $\mathcal{D}_0$, initial meta-instruction $\mathcal{I}_0$.
1: Initialize the exemplar set $\mathcal{E}_0 = \emptyset$, and the subset $\mathcal{E}'_0 = \emptyset$, meta-prompt-score set $\mathcal{S}_0 = \emptyset$, and cumulative score estimates $\hat{s}_i^{(0)} = 0$ for all $i \in \{1, \ldots, k\}$.
2: for iteration $t = 0, 1, \ldots, T-1$ do
3:     Construct meta-prompt $\mathcal{Q}_t = (\mathcal{D}_t, \mathcal{I}_t, \mathcal{E}'_t)$.
4:     Query the LLM $f(\cdot)$ using the meta-prompt $\mathcal{Q}_t$ to select the next action $x_t$: $x_t = f(\mathcal{Q}_t)$.
5:     Observe the score $s_t$ for $x_t$ using the task-specific evaluator: $s_t = \xi(x_t)$.
6:     Update the exemplar set $\mathcal{E}_{t+1} = \mathcal{E}_t \cup \{(x_t, s_t)\}$.
7:     Update the meta-prompt-score set $\mathcal{S}_{t+1} = \mathcal{S}_t \cup \{(\left[g(\mathcal{D}_t) \oplus g(\mathcal{I}_t)\right], s_t)\}$, where $g(\cdot)$ denotes the embedding function and $\oplus$ denotes concatenation.
8:     Update the parameters $\theta$ of the neural network (NN) $\mathcal{M}(g(\cdot); \theta)$ by using the updated $\mathcal{S}_{t+1}$ as the training set to minimize the MSE loss, yielding $\theta_{t+1}$.
9:     Update the cumulative score estimates $\hat{s}_i^{(t)}$ for all arms $i$ using the predicted scores from $\mathcal{M}(g(\cdot); \theta_{t+1})$:
           $\hat{s}_i^{(t+1)} = \hat{s}_i^{(t)} + \mathcal{M}(\left[g(\mathcal{D}_i) \oplus g(\mathcal{I}_i)\right]; \theta_{t+1})$
           $\forall i \in \{1, \ldots, k\}.$    (1)
10:    Compute the sampling distribution $P_t$ over all arms:
           $P_t[i] = \frac{\exp(\eta \hat{s}_i^{(t+1)})}{\sum_{j=1}^{k} \exp(\eta \hat{s}_j^{(t+1)})}, \quad \forall i \in \{1, \ldots, k\}$    (2)
11:    Sample an arm (i.e., the combination of a task description and a meta-instruction) from $P_t$: $(\mathcal{D}_{t+1}, \mathcal{I}_{t+1}) \sim P_t$.
12:    Select a sequence of exemplars $\mathcal{E}'_{t+1}$ from $\mathcal{E}_{t+1}$ following a pre-defined heuristic method.
```

Exploitation vs. Exploration. Our EXPO algorithm is able to achieve a principled balance between exploitation and exploration. The use of powerful pre-trained embedding and NNs allows us to achieve accurate score estimates. Therefore, the cumulative score estimate $\hat{s}_i^{(t+1)}$ (line 9 of Algo. 1) provides a reliable assessment of the quality of every arm i (i.e., every combination of task description and meta-instruction). This ensures that an arm with a large score is given a large weight in the sampling distribution P_t (line 10) and hence leads to reliable *exploitation*. Meanwhile, the inherent randomness in our randomized arm selection strategy ensures that enough *exploration* is performed in the domain of meta-prompts.

Batch Action Selection. In the description of our EXPO (Algo. 1), although we select one action x_t in every iteration t, this can be easily generalized to select a batch of actions. For example, when applying our EXPO to improve OPRO (Yang et al., 2024) (Sec. 4.1), we follow the practice of OPRO to select a batch of 8 actions/queries in every iteration (i.e., step 4 of Algo. 1) and set the temperature of the LLM to 1 to ensure the diversity of the selected actions. In order to obtain a noiseless and reliable score to assess the quality of the meta-prompt $\mathcal{Q}_t$, we set the temperature to 0 when selecting the last action and use its corresponding observed score as the score s_t of $\mathcal{Q}_t$ (line 5 of Algo. 1).

3.2 EXPO WITH EXEMPLAR SELECTION (EXPO-ES)

Previous works on LLM-based for sequential decision making often select the sequence of exemplars $\mathcal{E}'_{t+1}$ included in the meta-prompt $\mathcal{Q}_t$ using a fixed pre-defined heuristic (line 12 of Algo. 1). For example, OPRO includes the 20 exemplars with the highest observed scores in the meta-prompt, arranging them in descending order based on their scores (Yang et al., 2024); the LLM-based MAB method from Krishnamurthy et al. (2024) either includes all exemplars (ordered by their iteration sequence) in the prompt or includes a summarized representation of all exemplars. However, numerous previous works have reported that both the subset of exemplars and their ordering have significant impacts on the performance of LLM (Wu et al., 2024). Therefore, here we further extend our EXPO (Algo. 1) to additionally optimize the sequence of exemplars $\mathcal{E}'_{t+1}$ (i.e., to replace line



Linear Regression ($w = 2, b = 30$)

Linear Regression ($w = 36, b = -1$)

TSP (10 Nodes)

TSP (15 Nodes)

TSP (20 Nodes)

Figure 2: Results of different algorithms (mean ± standard error) in the Linear Regression and TSP task (Sec. 4.1). Lower is better.

12 of Algo. 1 by an automated method to select $\mathcal{E}'_{t+1}$), hence introducing our EXPO-ES algorithm (Algo. 2, App. A).

As a result of the dynamically changing task description and meta-instruction, the optimization of exemplar sequences becomes non-stationary as well. Therefore, we also dynamically optimize the exemplar sequence based on the EXP3 algorithm for adversarial bandits. That is, in every iteration of our EXPO-ES algorithm (Algo. 2), we firstly optimize the task description and meta-instruction (i.e., following lines 3-11 of Algo. 1), and then optimize the exemplar sequence $\mathcal{E}'_{t+1}$ in a similar way to Algo. 1.

Details of EXPO-ES (Algo. 2). Specifically, after the task description and meta-instruction are optimized (i.e., after lines 3-11 of Algo. 1), we firstly extract the embedding of the exemplar sequence $\mathcal{E}'_t$ used in this iteration: $g(\mathcal{E}'_t)$, and add $\left(g(\mathcal{E}'_t), s_t\right)$ to the *exemplar training set* $\mathcal{T}_{t+1}$ (line 4 of Algo. 2). Next, the updated dataset $\mathcal{T}_{t+1}$ is used to train an NN with parameters θ^{ES}_{t+1} (line 5), which is able to *estimate the score of any exemplar sequence*. Subsequently, we randomly sample k^{ES} exemplars sequences, each containing $\mathcal{L}$ exemplars, to be used as our *domain of exemplar sequences* (line 8). Next, for every candidate exemplar sequence in the domain, we need to obtain its cumulative score estimate (similar to line 9 of Algo. 1). Unfortunately, due to *the time-varying nature of the domain of exemplar sequences* (due to the addition of new exemplars and random sampling of exemplar sequences), we are no longer able to constantly maintain a cumulative score estimate for every exemplar sequence (i.e., arm) and update it in an incremental way. To this end, we save the parameters of the trained NN in every iteration in history; then for each sampled exemplar sequence in the domain, we obtain its score estimates from all NNs in the history and use their sum as the *cumulative score estimate* for this exemplar sequence (lines 9-14 of Algo. 2). Next, the cumulative score estimates for all exemplar sequences are used to compute the sampling distribution, from which the next exemplar sequence $\mathcal{E}'_{t+1}$ is sampled and used to the meta-prompt in the next iteration (lines 15-16).

4 Experiments

We firstly apply our algorithms to improve the performance of OPRO in the Linear Regression (LR) and traveling salesman problem (TSP) tasks, adopting the same experimental setting as Yang et al. (2024) (Sec. 4.1). Then, we use our algorithms to enhance the performance of the LLM-based MAB algorithm from Krishnamurthy et al. (2024).

4.1 Linear Regression and Traveling Salesman Problem

For both tasks here, we adopt GPT-3.5-Turbo as the LLM.

Linear Regression (LR). In the LR task, our goal is to find the optimal LR coefficients, w and b, that best fit a set of given noisy observations. We firstly choose the groundtruth LR coefficients w_{true} and b_{true}, and use them to generate noisy observations for 50 inputs x which are randomly and uniformly selected within $[-1, 1]$. Specifically, for each input x, we generate its noisy observation as $y = w_{\text{true}}x + b_{\text{true}} + \epsilon$ where ϵ is a Gaussian noise. We adopt the two most challenging choices of coefficients from Yang et al. (2024): (1) $w_{\text{true}} = 2, b_{\text{true}} = 30$ and (2) $w_{\text{true}} = 36, b_{\text{true}} = -1$. In this task, OPRO aims to find the optimal w and b which minimizes the regression error (i.e., mean squared error).

OPRO

Now you will help me minimize a function with two input variables w, b. I have some (w, b) pairs and the function values at those points. The pairs are arranged in descending order based on their function values, where lower values are better.

{EXEMPLARS}

Give me a new (w, b) pair that is different from all pairs above, and has a function value lower than any of the above. Do not write code. The output must end with a pair $[w, b]$, where w and b are numerical values.

EXPO

We will collaborate to optimize a function involving two parameters, \(w\) and \(b\). I possess a set of data points, each consisting of \((w, b)\) pairs and their corresponding function values. These pairs are systematically organized in reverse order, starting from the greatest to the smallest function values. Essentially, the lower the function value, the more optimal or preferable the pair. Consequently, our goal is to identify and analyze the \((w, b)\) pair that manifests the lowest function value, as this represents the perspective of optimum efficacy.

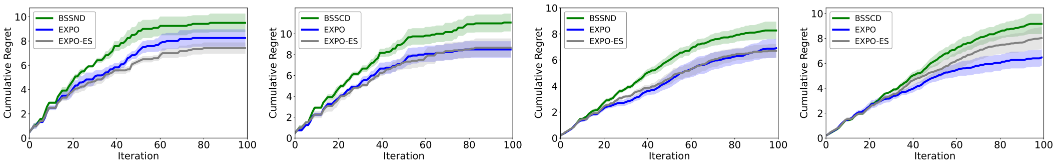
{EXEMPLARS}

To enhance the quality and expand on the existing instructions, follow these improved guidelines vis-à-vis designing a new and distinctive numerical pair: ensure the selected (w, b) combination diverges from prior examples and secures a function output lower than preceding values. Key details on methodology or calculations are not required—just ensure clarity in presenting a returned value that closes with the specific format $[w, b]$, where both w and b are distinct numerical figures.

Figure 3: The task description and meta-instruction used by OPRO (left) and optimized by our EXPO (right) in a Linear Regression task.

Traveling Salesman Problem (TSP). In the classical TSP problem (Jünger et al., 1995), given a set of n nodes with their coordinates, the objective is to find the shortest route that starts from a given node, traverses all nodes exactly once, and finally returns to the starting node. Therefore, our goal is to solve a discrete optimization problem in which the input variable is a trajectory and the goal is to minimize the total distance of the trajectory. We adopt TSP instances with 10, 15, and 20 randomly generated nodes, respectively, which represent increasing levels of difficulty.

The results for both tasks are shown in Fig. 2, which plot the regression error (i.e., mean squared error) for the LR tasks and optimality gap (i.e., the difference between the total distance of the discovered route and that of the optimal route) for the TSP tasks (lower is better for both tasks). Of note, in addition to the standard OPRO (pink curves) (Yang et al., 2024), we have also proposed an enhanced variant of OPRO (green curves) in which we added some further clarifications to the task description (see App. B.2.5 for more details). The enhanced variant consistently improves the performance of the standard OPRO (Fig. 2).[2] More importantly, the results in Fig. 2 show that **in all tasks, our EXPO algorithm (blue curves) significantly and consistently outperforms OPRO**, including both standard OPRO and its enhanced variant. This demonstrates that our meta-prompt optimization approach, grounded in adversarial bandits, leads to more efficient (i.e., faster convergence) and more effective (i.e., improved final performance) LLM-based sequential decision-making.



BSSND (easy) BSSCD (easy) BSSND (hard) BSSCD (hard)

Figure 4: Cumulative regret of different algorithms in the LLM-based MAB experiments (Sec. 4.2). Lower is better.

Meanwhile, our EXPO-ES algorithm, which is additionally equipped with automated exemplar selection, considerably improves the performance of EXPO in the LR tasks yet performs on par with EXPO in the TSP tasks. This is likely because the exemplars play a more important role in the LR tasks than the TSP tasks. Specifically, in LR, *the input-output exemplars provide important information for identifying the optimal LR coefficients* (Wu et al., 2024). Therefore, selecting better exemplars (via our EXPO-ES) brings significant performance boost. On the other hand, in the TSP tasks, due to the challenging nature of the tasks, it is difficult for the LLM to infer crucial and useful information from the exemplars. Therefore, the other components in the meta-prompt (i.e., the task

[2] As discussed in the last paragraph of Sec. 3.1, we have slightly modified OPRO to select the last action in the batch using a temperature of 0. We empirically show that this leads to comparable performance with the original OPRO which uses a temperature of 1 to choose all 8 actions (see Fig. 14 in App. C.2).

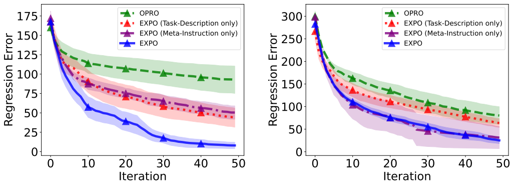
description and meta-instruction) provide more useful information in the TSP tasks. As a result, *selecting better exemplars does not lead to noticeable performance gains in the TSP tasks*. Fig. 3 provides an illustration of the original task description and meta-instruction used by OPRO and those discovered by our `EXPO` algorithm for the LR tasks, whereas the corresponding meta-prompts for the TSP tasks are displayed in Fig. 16 in App. C.4.

4.2 LLM-Based Multi-Armed Bandits (MAB)

The work of Krishnamurthy et al. (2024) has used an LLM to sequentially select the arms/actions in MAB and proposed methods to manually design the meta-prompt. Their prompt design consists of 5 components with each having 2 possible choices, which gives rise to a total of $2^5 = 32$ possible prompts. Here we show that our algorithms can be used to automatically optimize their manually designed prompts to further enhance their performance. Specifically, we adopt 2 of their prompt designs: BSSND and BSSCD, and apply our `EXPO` and `EXPO-ES` algorithms to optimize the important components in these prompt designs. Following Krishnamurthy et al. (2024), we use two MAB instances: *easy* and *hard*. We adopt GPT-4-Turbo as the LLM here. More details on the experimental design are deferred to App. B.3.1. The results for the 4 experimental settings (i.e., 2 prompt designs × 2 MAB instances) are shown in Fig. 4, which demonstrate that our `EXPO` and `EXPO-ES` algorithms are able to significantly reduce the cumulative regret of MAB in this task across different prompt desings and MAB instances. We illustrate the comparison between the original meta-prompt and the one optimized by our `EXPO` in Figs. 17 and 18 in App. C.4.

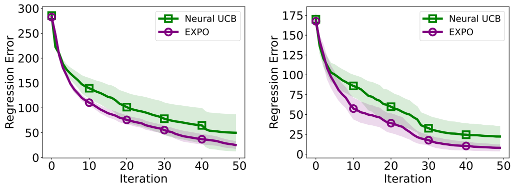
5 Ablation Study

Only Optimizing Task Description or Meta-Instruction. Our `EXPO` jointly optimize the task description $\mathcal{D}$ and the meta-instruction $\mathcal{I}$. Here we evaluate the performance of optimizing either $\mathcal{D}$ or $\mathcal{I}$ alone. The results in Fig. 5 show that jointly optimizing them indeed leads to significantly better performance. However, optimizing these components alone still consistently outperforms OPRO.



$w = 2,\ b = 30$ $w = 36,\ b = -1$

Figure 5: Results of our `EXPO` when only optimizing the task description or the meta-instruction.



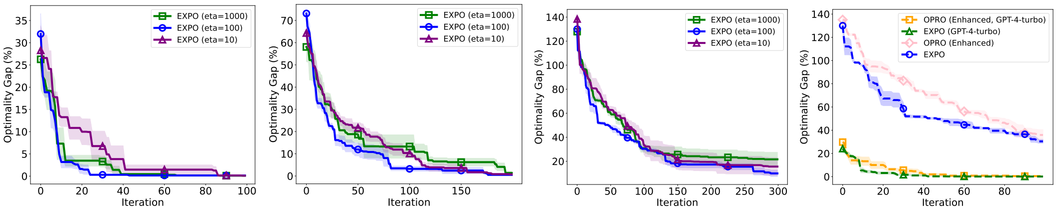
Linear Regression ($w = 36,\ b = -1$)

Linear Regression ($w = 2,\ b = 30$)

Figure 6: Comparison of our `EXPO` with NeuralUCB (i.e., a representative stochastic MAB algorithm) in the LR tasks.

Comparison with Stochastic MAB Algorithms: Upper Confidence Bound. Classical stochastic MAB algorithms, such as those based on upper confidence bound (UCB), have been applied to prompt optimization in a number of recent works (Lin et al., 2024a;b; Wu et al., 2024) and yielded strong performance. However, as we have discussed in Sec. 1, in meta-prompt optimization for LLM-based sequential decision-making, the non-stationary reward observations render these stochastic MAB methods unsuitable. Here we verify this by comparing our `EXPO` with the *NeuralUCB* algorithm adopted by Lin et al. (2024b); Wu et al. (2024). The results for the Linear Regression tasks are displayed in Fig. 6, which show that NeuralUCB indeed significantly underperforms in the problem of meta-prompt optimization for LLM-based agents. The results for the TSP tasks are consistent with the results here (Fig. 19 in App. C.5). These results provide further justifications for our proposed adversarial bandit-based algorithms.

Impact of the Degree of Exploration. Here we examine the impact of the degree of exploration, i.e., the value of η (see line 10 of Algo. 1). The results (Fig. 7) show that an excessively large degree of exploration (i.e., a small $\eta = 10$) or an overly small degree of exploration (i.e., a large $\eta = 1000$) both deteriorate the performance. Moreover, the results also demonstrate that in easier tasks (i.e., TSP with 10 nodes), imposing a smaller degree of exploration (i.e., $\eta = 1000$) leads to better performance



TSP
(10 nodes)

TSP
(15 nodes)

TSP
(20 nodes)

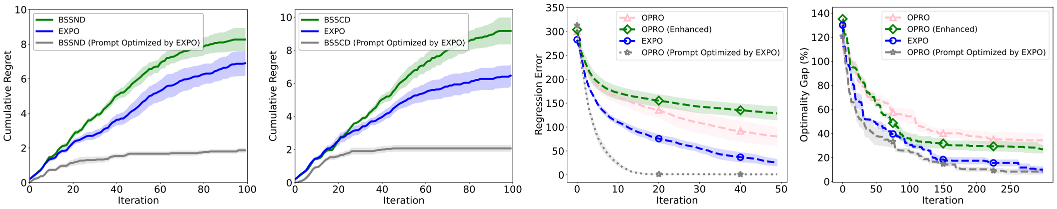
TSP
(20 Nodes, GPT-4-Turbo)

Figure 7: First three figures: ablation study on impact of exploration parameter η. Rightmost figure: results using GPT-4-Turbo.

compared to $\eta = 10$, because it allows our EXPO to quickly converge to the optimal solution. On the other hand, in more challenging tasks (i.e., TSP with 20 nodes), more exploration (i.e., $\eta = 10$) results in better performance (than $\eta = 1000$), because it makes it easier for our EXPO to escape local optimum.

Experiments With Other LLMs. To evaluate the effectiveness of our approach when combined with different LLMs, here we adopt the challenging TSP task with 20 nodes and replace the GPT-3.5-Turbo model used in our original experiments (Sec. 4.1) by the more advanced GPT-4-Turbo model. The results in Fig. 7 (rightmost figure) show that the use of the more advanced GPT-4-Turbo model significantly improves the performance of both OPRO and our EXPO. More importantly, as visualized more clearly in Fig. 13 in App. C.1, when both adopting GPT-4-Turbo, our EXPO still significantly outperforms OPRO. The results show that our EXPO can effectively improve the performance of LLM-based agents across different LLMs.

Effectiveness of the Optimal Prompt Discovered by EXPO. To further verify the ability of our EXPO to identify effective meta-prompts, here we replace the original task description and meta-instruction in an LLM-based sequential decision-making algorithm (e.g., OPRO) by the optimal ones discovered by our EXPO. For example, for ORPO, we firstly run our EXPO to completion, and then use the final meta-prompt selected by our EXPO as the meta-prompt to execute OPRO again. The results in Fig. 8 show that fixing the meta-prompt to be the one optimized by our EXPO leads to dramatic performance boost to LLM-based sequential decision-making.



BSSND (hard)

BSSCD (hard)

Linear Regression
$(w = 36,\ b = -1)$

TSP
(20 Nodes)

Figure 8: Results achieved by fixing the meta-prompt to be the optimal one discovered by our EXPO (gray curves).

6 Related Work

Prompt Optimization. The field of prompt optimization has been gaining significant popularity recently. Earlier works on this topic have focused on optimizing the prompt for white-box LLMs Shin et al. (2020); Shi et al. (2023); Lester et al. (2021); Li & Liang (2021); Zhong et al. (2021); Deng et al. (2022). More recently, a number of works have developed prompt optimization methods for black-box LLMs Chen et al. (2023); Zhou et al. (2023); Fernando et al. (2023); Guo et al. (2024); Hu et al. (2024); Lin et al. (2024b); Zhan et al. (2024); Juneja et al. (2024); Wang et al. (2023b); Kong et al. (2024); Schneider et al. (2024); Shi et al. (2024). In addition, some recent works have focused on automatically selecting the exemplars for in-context learning Wang et al. (2023a); Chang & Jia (2023); Li & Qiu (2023); Zhang et al. (2022); Nguyen & Wong (2023); Albalak et al. (2024); Ye et al. (2023); Liu et al. (2022); Gao et al. (2024); Rubin et al. (2022); Ye et al. (2023); Levy et al. (2023);

Gupta et al. (2023), whereas a few methods have been proposed to jointly optimize the prompt and select the exemplars Opsahl-Ong et al. (2024); Wan et al. (2024); Wu et al. (2024). However, to the best of our knowledge, our algorithm is the first approach that is able to efficiently optimize the meta-prompt for LLM-based agents in sequential decision-making tasks.

LLM-Based Sequential Decision-Making. Some recent works have proposed to leverage the strong capability of LLMs to solve sequential decision-making tasks, such as Bayesian optimization Yang et al. (2024), multi-armed bandits Krishnamurthy et al. (2024); Xia et al. (2024); Chen et al. (2024); Mukherjee et al. (2024), and reinforcement learning Dai et al. (2024); Monea et al. (2024); Wang et al. (2024a). However, these works often provide a fixed manually designed meta-prompt to the LLM, and are hence unable to fully unleash the potential of LLM-based sequential decision-making. The field of LLM-based agents has seen a surging interest recently, for which a number of benchmarks have been proposed Liu et al. (2023); Wu et al. (2023); Xi et al. (2024). We defer more a comprehensive discussion of LLM-based agents to recent surveys on this topic Cheng et al. (2024); Wang et al. (2024b); Xi et al. (2023).

7 Conclusion

In this work, we have proposed our `EXPO` algorithm to automatically optimize the meta-prompt for LLM-based sequential decision-making tasks. We further extend our `EXPO` to derive the `EXPO-ES` algorithm, which additionally optimizes the exemplars in the meta-prompt. Our algorithms use neural networks to estimate the scores of different meta-prompts and sequentially selects the meta-prompts in a randomized fashion based on adversarial bandits. We use extensive experiments to show that our algorithms considerably and consistently improve the performance of LLM-based sequential decision-making.

References

Alon Albalak, Yanai Elazar, Sang Michael Xie, Shayne Longpre, Nathan Lambert, Xinyi Wang, Niklas Muennighoff, Bairu Hou, Liangming Pan, Haewon Jeong, Colin Raffel, Shiyu Chang, Tatsunori Hashimoto, and William Yang Wang. A survey on data selection for language models. *arXiv:2402.16827*, 2024.

Ting-Yun Chang and Robin Jia. Data curation alone can stabilize in-context learning. In *Proc. ACL*, pp. 8123–8144, 2023.

Dingyang Chen, Qi Zhang, and Yinglun Zhu. Efficient sequential decision making with large language models. *arXiv preprint arXiv:2406.12125*, 2024.

Lichang Chen, Jiuhai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. InstructZero: Efficient instruction optimization for black-box large language models. *arXiv:2306.03082*, 2023.

Yuheng Cheng, Ceyao Zhang, Zhengwen Zhang, Xiangrui Meng, Sirui Hong, Wenhao Li, Zihao Wang, Zekai Wang, Feng Yin, Junhua Zhao, et al. Exploring large language model based intelligent agents: Definitions, methods, and prospects. *arXiv preprint arXiv:2401.03428*, 2024.

Zhenwen Dai, Federico Tomasi, and Sina Ghiassian. In-context exploration-exploitation for reinforcement learning. *arXiv preprint arXiv:2403.06826*, 2024.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric Xing, and Zhiting Hu. RLPrompt: Optimizing discrete text prompts with reinforcement learning. In *Proc. EMNLP*, pp. 3369–3391, 2022.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv:2309.16797*, 2023.

Lingyu Gao, Aditi Chaudhary, Krishna Srinivasan, Kazuma Hashimoto, Karthik Raman, and Michael Bendersky. Ambiguity-aware in-context learning with large language models. *arXiv:2309.07900*, 2024.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *Proc. ICLR*, 2024.

Shivanshu Gupta, Matt Gardner, and Sameer Singh. Coverage-based example selection for in-context learning. In *Proc. EMNLP*, pp. 13924–13950, 2023.

Wenyang Hu, Yao Shu, Zongmin Yu, Zhaoxuan Wu, Xiangqiang Lin, Zhongxiang Dai, See-Kiong Ng, and Bryan Kian Hsiang Low. Localized zeroth-order prompt optimization. In *Proc. NeurIPS*, 2024.

Gurusha Juneja, Nagarajan Natarajan, Hua Li, Jian Jiao, and Amit Sharma. Task facet learning: A structured approach to prompt optimization. *arXiv preprint arXiv:2406.10504*, 2024.

Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. The traveling salesman problem. *Handbooks in operations research and management science*, 7:225–330, 1995.

Weize Kong, Spurthi Amba Hombaiah, Mingyang Zhang, Qiaozhu Mei, and Michael Bendersky. Prewrite: Prompt rewriting with reinforcement learning. *arXiv preprint arXiv:2401.08189*, 2024.

Akshay Krishnamurthy, Keegan Harris, Dylan J Foster, Cyril Zhang, and Aleksandrs Slivkins. Can large language models explore in-context? *arXiv preprint arXiv:2403.15371*, 2024.

Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proc. EMNLP*, pp. 3045–3059, 2021.

Itay Levy, Ben Bogin, and Jonathan Berant. Diverse demonstrations improve in-context compositional generalization. In *Proc. ACL*, pp. 1401–1422, 2023.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597, 2021.

Xiaonan Li and Xipeng Qiu. Finding support examples for in-context learning. In *Proc. EMNLP*, pp. 6219–6235, 2023.

Xiaoqiang Lin, Zhongxiang Dai, Arun Verma, See-Kiong Ng, Patrick Jaillet, and Bryan Kian Hsiang Low. Prompt optimization with human feedback. *arXiv preprint arXiv:2405.17346*, 2024a.

Xiaoqiang Lin, Zhaoxuan Wu, Zhongxiang Dai, Wenyang Hu, Yao Shu, See-Kiong Ng, Patrick Jaillet, and Bryan Kian Hsiang Low. Use your INSTINCT: Instruction optimization using neural bandits coupled with transformers. In *Proc. ICML*, 2024b.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for GPT-3? In *Proc. DeeLIO: Deep Learning Inside Out*, pp. 100–114, 2022.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *Proc. ACL*, pp. 8086–8098, 2022.

Giovanni Monea, Antoine Bosselut, Kianté Brantley, and Yoav Artzi. Llms are in-context reinforcement learners. 2024.

Subhojyoti Mukherjee, Josiah P Hanna, Qiaomin Xie, and Robert Nowak. Pretraining decision transformers with reward prediction for in-context multi-task structured bandit learning. *arXiv preprint arXiv:2406.05064*, 2024.

Tai Nguyen and Eric Wong. In-context example selection with influences. *arXiv:2302.11042*, 2023.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. *arXiv preprint arXiv:2406.11695*, 2024.

Ohad Rubin, Jonathan Herzig, and Jonathan Berant. Learning to retrieve prompts for in-context learning. In *Proc. NAACL*, pp. 2655–2671, 2022.

Lennart Schneider, Martin Wistuba, Aaron Klein, Jacek Golebiowski, Giovanni Zappella, and Felice Antonio Merra. Hyperband-based bayesian optimization for black-box prompt selection. *arXiv preprint arXiv:2412.07820*, 2024.

Chengshuai Shi, Kun Yang, Jing Yang, and Cong Shen. Best arm identification for prompt learning under a limited budget. *arXiv preprint arXiv:2402.09723*, 2024.

Weijia Shi, Xiaochuang Han, Hila Gonen, Ari Holtzman, Yulia Tsvetkov, and Luke Zettlemoyer. Toward human readable prompt tuning: Kubrick's the shining is a good movie, and a good prompt too? In *Proc. EMNLP*, pp. 10994–11005, 2023.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. Eliciting knowledge from language models using automatically generated prompts. In *Proc. EMNLP*, pp. 4222–4235, 2020.

Xingchen Wan, Ruoxi Sun, Hootan Nakhost, and Sercan O Arik. Teach better or show smarter? on instructions and exemplars in automatic prompt optimization. *arXiv preprint arXiv:2406.15708*, 2024.

Jiuqi Wang, Ethan Blaser, Hadi Daneshmand, and Shangtong Zhang. Transformers learn temporal difference methods for in-context reinforcement learning. *arXiv preprint arXiv:2405.13861*, 2024a.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024b.

Xinyi Wang, Wanrong Zhu, Michael Saxon, Mark Steyvers, and William Yang Wang. Large language models are latent variable models: Explaining and finding good demonstrations for in-context learning. In *Proc. NeurIPS*, pp. 15614–15638, 2023a.

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

Yue Wu, Xuan Tang, Tom M Mitchell, and Yuanzhi Li. Smartplay: A benchmark for llms as intelligent agents. *arXiv preprint arXiv:2310.01557*, 2023.

Zhaoxuan Wu, Xiaoqiang Lin, Zhongxiang Dai, Wenyang Hu, Yao Shu, See-Kiong Ng, Patrick Jaillet, and Bryan Kian Hsiang Low. Prompt optimization with EASE? efficient ordering-aware automated selection of exemplars. In *Proc. NeurIPS*, 2024.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*, 2023.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, et al. Agentgym: Evolving large language model-based agents across diverse environments. *arXiv preprint arXiv:2406.04151*, 2024.

Fanzeng Xia, Hao Liu, Yisong Yue, and Tongxin Li. Beyond numeric awards: In-context dueling bandits with llm agents. *arXiv preprint arXiv:2407.01887*, 2024.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *Proc. ICLR*, 2024.

Jiacheng Ye, Zhiyong Wu, Jiangtao Feng, Tao Yu, and Lingpeng Kong. Compositional exemplars for in-context learning. In *Proc. ICML*, pp. 39818–39833, 2023.

Heshen Zhan, Congliang Chen, Tian Ding, Ziniu Li, and Ruoyu Sun. Unlocking black-box prompt tuning efficiency via zeroth-order optimization. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 14825–14838, 2024.

Yiming Zhang, Shi Feng, and Chenhao Tan. Active example selection for in-context learning. In *Proc. EMNLP*, pp. 9134–9148, 2022.

Zexuan Zhong, Dan Friedman, and Danqi Chen. Factual probing is [MASK]: Learning vs. learning to recall. In *Proc. NAACL*, pp. 5017–5033, 2021.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *Proc. ICLR*, 2023.

A Our EXPO-ES Algorithm to Additionally Optimize the Exemplar Sequences

Our complete EXPO-ES algorithm is described in Algo. 2. As we have discussed in Sec. 3.2, there are two major differences compared to the way in which our EXPO algorithm optimizes the task description and meta-instruction (Algo. 1). Firstly, our domain of k^{ES} arms (i.e., every arm corresponds to a randomly sampled exemplar sequence) changes in every iteration (line 8). Secondly, as a result of the time-varying domains, we need to save a copy of the parameters of the NN trained in every iteration in order to compute the cumulative score estimates (lines 9-14).

Simplified Variant of Our EXPO-ES Algorithm. When applying our EXPO-ES algorithm to the LLM-based MAB algorithm in Krishnamurthy et al. (2024) (Sec. 4.2), we have adopted a simplified variant of our EXPO-ES. This is because in the problem setting from Krishnamurthy et al. (2024), the number of arms is small. Therefore, instead of including a subset of the history of exemplars in the prompt, their algorithm has instead included a summarized observation history. An example of such summarized observation history with 5 arms (represented by 5 buttons with different colors) is given in Fig. 9 below. Therefore, here we aim to optimize the format of the summarized observation history. Specifically, we *optimize the order of the arms* in the summarized history, and our domain of arms consist of *all cyclically shifted variants of the following sequence of buttons*: $\{$blue button, green button, red button, yellow button, purple button$\}$. For example, some other arms (button sequences) in our domain include: $\{$green button, red button, yellow button, purple button, blue button$\}$ and $\{$red button, yellow button, purple button, blue button, green button$\}$. As a result, unlike our original EXPO-ES algorithm described in Sec. 3.2, here we do not suffer from the issue of *time-varying domain of arms*.

```
...

blue button: pressed 2 times with average reward 0.5
green button: pressed 1 times with average reward 0.0
red button: pressed 1 times with average reward 1.0
yellow button: pressed 0 times
purple button: pressed 1 times with average reward 0.0

...
```

Figure 9: An example of the summarized observation history used by the LLM-based MAB algorithm from Krishnamurthy et al. (2024).

Algorithm 2 EXPO-ES

```
input Initial task description $\mathcal{D}_0$, initial meta-instruction $\mathcal{I}_0$.
    Maximum number $\mathcal{L}$ of exemplars in the meta-prompt, the number $k^{\text{ES}}$ of exemplar sequences in the domain.
Initialize the exemplar set $\mathcal{E}_0 = \emptyset$, and the subset $\mathcal{E}'_0 = \emptyset$, meta-prompt-score set $\mathcal{S}_0 = \emptyset$, and cumulative score estimates $\hat{s}_j^{(0)}$ for all $j \in \{1, \ldots, k^{\text{ES}}\}$.
    Initialize the history of NN parameters $\Theta_{\text{history}} = \emptyset$, and the exemplar training set $\mathcal{T}_0 = \emptyset$.
for iteration $t = 0, 1, \ldots, T-1$ do
    Lines 3-11 of Algo. 1.
    Compute the embedding $g(\mathcal{E}'_t)$ of the selected exemplar sequence $\mathcal{E}'_t$, and add $g(\mathcal{E}'_t)$ and its score $s_t$ to the exemplar training set: $\mathcal{T}_{t+1} \leftarrow \mathcal{T}_t \cup \{(g(\mathcal{E}'_t), s_t)\}$.
    Update the parameters $\theta^{\text{ES}}$ of the NN $\mathcal{M}_{\text{ES}}(g(\cdot); \theta^{\text{ES}})$ by using the updated $\mathcal{T}_{t+1}$ as the training set to minimize the MSE loss, yielding $\theta^{\text{ES}}_{t+1}$.
    Add the updated parameters to the history: $\Theta_{\text{history}} \leftarrow \Theta_{\text{history}} \cup \{\theta^{\text{ES}}_{t+1}\}$.
    if $|\mathcal{E}_{t+1}| > \mathcal{L}$ then
        Randomly generate $k^{\text{ES}}$ sequences of $\mathcal{L}$ exemplars from the exemplar set $\mathcal{E}_{t+1}$: $\{\mathcal{E}^1_{t+1}, \mathcal{E}^2_{t+1}, \ldots, \mathcal{E}^{k^{\text{ES}}}_{t+1}\}$, in which every $\mathcal{E}^j_{t+1}$ represents an ordered set of $\mathcal{L}$ exemplars from $\mathcal{E}_{t+1}$.
        Initialize cumulative score estimates $\hat{s}_j^{(0)} = 0$ for all $j \in \{1, \ldots, k^{\text{ES}}\}$.
        for each $\mathcal{E}^j_{t+1}$ in $\{\mathcal{E}^1_{t+1}, \ldots, \mathcal{E}^{k^{\text{ES}}}_{t+1}\}$ do
            Initialize cumulative score $\hat{s}_j^{(0)} = 0$.                                    cumulative
            for each historical model parameter $\theta^{\text{ES}}_i \in \Theta_{\text{history}}$ do    } score
                Update the cumulative score for $\mathcal{E}^j_{t+1}$: $\hat{s}_j^{(i)} = \hat{s}_j^{(i-1)} + \mathcal{M}_{\text{ES}}(g(\mathcal{E}^j_{t+1}); \theta^{\text{ES}}_i)$.    estimates
        Compute the final cumulative score estimates: $\hat{s}_j^{(\text{final})} = \hat{s}_j^{(|\Theta_{\text{history}}|)}, \quad \forall j \in \{1, \ldots, k^{\text{ES}}\}$.
        Compute the sampling distribution $P^{\text{ES}}_t$ over the $k$ exemplar sequences:
            $$P^{\text{ES}}_t[j] = \frac{\exp(\eta \hat{s}_j^{(\text{final})})}{\sum_{l=1}^{k^{\text{ES}}} \exp(\eta \hat{s}_l^{(\text{final})})}, \quad \forall j \in \{1, \ldots, k^{\text{ES}}\}. \qquad (5)$$
        Sample an exemplar sequence $\mathcal{E}'_{t+1} \sim P^{\text{ES}}_t$.
```

Therefore, when applying our EXPO-ES algorithm to improve the LLM-based MAB method from Krishnamurthy et al. (2024) (Sec. 4.2), we make two modifications to our standard EXPO-ES algorithm described in Algo. 2. Firstly, instead of randomly sampling k^{ES} exemplar sequences to form our domain of exemplar sequences, here our domain remains fixed across different iterations, i.e., all cyclically shifted variants of the arms. Secondly, since here we do not suffer from the issue of time-varying domain of arms (i.e., exemplar sequences), we can resort to the incremental update of the cumulative reward estimates adopted by our EXPO algorithm (line 9 of Algo. 1). As a result, we do not need to save a copy of the parameters of the NN trained in every iteration.

B More Details on Our Experimental Settings

B.1 More Details on the Generation of the Domain of Task Description and Meta-Instruction

Here we describe the details about how we generate the domain of task descriptions and meta-instructions. Below we provide the prompt we have used to instruct the LLM to generate every prompt in the domain.

Example Query: Meta-Prompt Instruction Rephrasing Template

To achieve a more effective TASK description and INSTRUCTION and convey its core essence more clearly, please enhance the content in the quote by rephrasing and changing some information:
”{INITIAL_META-PROMPT}”
Please return directly the modified description without additional description.
The modified description:

Generation of the Domain. To effectively generate task-specific prompts, we utilized an initial prompt to guide the LLM in creating diverse task descriptions and meta-instructions. For each task, the LLM was prompted 100 times to rephrase the task description and meta-instruction separately, resulting in 100 unique rephrased prompts for each. Combined with the initial prompt, this process produced a total of 101×101 combinations of task descriptions and meta-instructions for each task.

To optimize computational efficiency, we pre-compute the embeddings of all task descriptions and meta-instructions in the domain using the embedding model $g(\cdot)$ and store the results to prevent redundant calculations during subsequent experiments.

For the rephrasing process, we employed the GPT-4 model with a temperature setting of 1.3, ensuring diverse and high-quality rephrased prompts for both task descriptions and meta-instructions.

B.2 More Details on OPRO for the Linear Regression and Traveling Salesman Problem (Sec. 4.1)

B.2.1 Task Setting.

Linear Regression. We conduct experiments on Linear Regression by selecting two challenging ground truth weight-bias (w, b) pairs. The experiments follow the OPRO framework, which requires warm-starting the LLM with initial exemplars. Using a fixed random seed, we first generate 50 random data points uniformly distributed within the range $[-1, 1]$, which perfectly satisfy the ground truth $w_{\text{true}}, b_{\text{true}}$ pairs, ensuring that these data points can serve as the foundation for evaluating the LLM's ability to model the relationships. Additionally, 5 w, b pairs with corresponding scores, sampled within the range $[10, 20]$, are generated using another fixed random seed to serve as the initial exemplars. At each iteration, the LLM is prompted 8 times (consisting of 1 inference with a temperature setting of $T = 0$ and 7 inferences with a temperature setting of $T = 1$) using the current exemplars, and the prompt is updated based on the generated outputs. The exemplars are dynamically updated to include the top 20 w, b pairs and their associated scores from all historical records across iterations, ensuring the LLM is always guided by the best-performing examples. The total number of iterations is set to 50, and each ground truth configuration is repeated 5 times for consistency.

Traveling Salesman Problem (TSP). For the TSP task, experiments are conducted on three problem sizes defined by the number of nodes: 10, 15, and 20. For each TSP instance, the problem is defined by randomly generating $n = 10, 15, 20$ nodes, where the x and y coordinates of each node are sampled uniformly from the range $[-100, 100]$. For each configuration, a specific TSP instance is generated using a fixed random seed, and a single random seed is used to generate warm-start exemplars to initialize the LLM prompts. To initialize the optimization process, we randomly sample 5 different TSP routes along with their corresponding total distances. These routes and their lengths are used as the initial exemplars for the LLM. Each iteration consists of 8 prompt calls to the LLM, followed by an update of the exemplars based on the generated results. More specifically, during each iteration, the GPT-3.5-turbo is prompted 8 times using the same prompt, consisting of 1 inference with a temperature setting of $T = 0$ to ensure stability and 7 inferences with a temperature setting of $T = 1$ to encourage exploration. Similar to the Linear Regression task, the exemplars for TSP are updated to include the top 20 historical solutions with the best scores, ensuring the prompt leverages the most effective examples. The number of iterations is set to 100, 200, and 300 for 10-node, 15-node, and 20-node TSP problems, respectively, to account for the increasing complexity of the tasks. Each node configuration is repeated 3 times to ensure consistency and reliability.

B.2.2 Evaluation Metrics.

Linear Regression. In the Linear Regression task, the performance of the algorithms is evaluated using the Mean Squared Error (MSE) metric. Given a set of N one-dimensional input data points $\mathbf{x} \in \mathbb{R}$ and their corresponding ground truth labels $\mathbf{y} \in \mathbb{R}$, the MSE is computed as:

$$\text{MSE} = \frac{1}{N} \left\| \mathbf{y} - (w \cdot \mathbf{x} + b) \right\|^2 ,$$

where $w \in \mathbb{R}$ and $b \in \mathbb{R}$ are the weight and bias parameters inferred by the LLM, and N is the total number of data points.

Traveling Salesman Problem (TSP). For the TSP task, the performance of the LLM-generated solutions is evaluated based on the total Euclidean distance of the TSP tour. Given a set of two-dimensional points $\{(x_i, y_i)\}_{i=1}^{N}$, where N is the total number of nodes, the length of a proposed TSP tour $\mathcal{P} = [\pi(1), \pi(2), \dots, \pi(N), \pi(1)]$ is computed as:

$$\text{Length} = \sum_{i=1}^{N} \sqrt{\left(x_{\pi(i+1)} - x_{\pi(i)}\right)^2 + \left(y_{\pi(i+1)} - y_{\pi(i)}\right)^2},$$

where π represents the permutation of nodes in the proposed tour, and $\pi(N+1) = \pi(1)$ ensures the tour returns to the starting node.

To evaluate the convergence and effectiveness of the agents, we use the *Optimality Gap* metric, which quantifies the deviation of the solver's best-found solution from the true optimal solution. It is defined as:

$$\text{Optimality Gap} = \frac{\text{SolverOptimal} - \text{Optima}}{\text{Optima}} \times 100\%,$$

where:

- SolverOptimal denotes the shortest tour length found by the solver up to the current iteration.
- Optima is the length of the known optimal TSP tour.

B.2.3 Design of Prompt Score.

In both the Linear Regression and TSP tasks, optimal solutions are characterized by lower evaluation scores. To align with the requirements of the algorithm and ensure more stable learning, we define the *Prompt Score* using the formula:

$$\text{Prompt Score} = \frac{-\text{Evaluation Score} + b}{b},$$

where $b > 0$ is a stabilizing constant. This formulation ensures that lower evaluation scores correspond to higher prompt scores, which better facilitates the optimization process and contributes to steady algorithmic learning.

For the Linear Regression task, the *Evaluation Score* is defined as the Mean Squared Error (MSE) of the weight-bias (w, b) pairs proposed by the algorithm at each iteration under a Temperature=0 stable inference. The MSE is computed based on the provided one-dimensional data points.

For the TSP task, the *Evaluation Score* corresponds to the total Euclidean distance (*Length*) of the TSP tour proposed by the algorithm at each iteration, also under a Temperature=0 stable inference.

B.2.4 Details about the Models and Parameters in Our Algorithms

LLM Agents and Embedding Model. In our experiments, the primary LLM agent used is GPT-3.5-Turbo. For embedding generation, we utilized OpenAI's `text-embedding-3-large` model, which outputs embeddings of dimensionality 3072. These embeddings were used to represent both the task description and meta-instruction in the `EXPO` framework. The embeddings were also employed to represent the exemplars in the `EXPO-ES` framework. During each iteration of inference, the LLM agent performed 1 prediction with a temperature setting of $T = 0$ to provide a stable solution and 7 additional predictions with a temperature setting of $T = 1$ to encourage exploration.

Neural Network Parameters. For the EXPO, the input to the neural network consists of the concatenated embeddings of the task description and meta-instruction, resulting in an input dimensionality of $3072 + 3072 = 6144$. The neural network employs a single hidden layer with a width of 1536 and produces a single scalar output. The training objective is to minimize the Mean Squared Error (MSE) loss function.

For the EXPO-ES, the exemplar selection process differs depending on the iteration count. During the initial iterations, when fewer than 20 optimal historical records are available, we use all available exemplars. As the iteration count increases, exemplars are selected from the top $\mathtt{min}(\text{total exemplar records}, 30)$ historical optimal records. From this pool, 257 exemplars are constructed, consisting of 256 randomly selected exemplars and 1 heuristic exemplar generated from a combination of 20 best historical records. The neural network for EXPO-ES operates on an input dimensionality of 3072, corresponding to the embedding of a single exemplar. It employs a single hidden layer with a width of 512 and produces a single scalar output. The training objective is to minimize the Mean Squared Error (MSE) loss.

EXP3 Learning Rate. In the EXPO , the learning rate parameter η_{desc} is set to 100 for selecting task descriptions and meta-instruction combinations. In the EXPO-ES, η_{desc} is also set to 100 for selecting task descriptions and meta-instruction combinations, while η_{exemplar} is set to 10 for selecting exemplars.

B.2.5 Enhanced OPRO

Here, we describe how we have enhanced the original algorithm Yang et al. (2024) by modifying its prompts.

During initial experiments with the meta-prompts provided by the original OPRO algorithm Yang et al. (2024) for task description rephrasing, we observed that the LLM often misinterprets the *descending order* semantics described in the original design. In tasks like TSP and Linear Regression, where better solutions correspond to lower evaluation scores, *descending order* is intended to arrange solutions from high evaluation scores to low. However, the LLM frequently misunderstands this as a *descending order of solution quality*, interpreting higher-ranked solutions as better and lower-ranked ones as worse, which is contrary to the intended meaning.

To address this issue, we enhance the orginal meta-prompts by explicitly clarifying the semantics of descending order in the context of evaluation scores. This modification ensures that the LLM accurately understand the intended instructions. When tested with the enhanced prompts, the problem was resolved, and the LLM is able to consistently generate correct rephrased task descriptions. For a clearer illustration, we provide below the original OPRO meta-prompt (Fig. 10) and our enhanced OPRO meta-prompt (Fig. 11).

The task description in the original OPRO prompt

You are given a list of points with coordinates below: {POINTS}.
Below are some previous traces and their lengths. The traces are arranged in descending order based on their lengths, where lower values are better.
......

Figure 10: The task description in the original OPRO prompt.

The task description in our enhanced OPRO prompt

You are given a list of points with coordinates below: {POINTS}.
Below are some previous traces and their lengths. The traces are arranged in descending order based on their lengths, where smaller lengths indicate better solutions. Therefore, the traces are listed from the largest length to the smallest, the trace with the smallest length is considered the most optimal.
......

Figure 11: The task description in the enhanced OPRO prompt. The texts we have modified are highlighted in red.

B.3 More Details on the LLM-Based Multi-Armed Bandits Task (Sec. 4.2)

B.3.1 Explanation of BSSCD and BSSND

We provide a detailed explanation and demonstration of prompt designs for both BSSCD and BSSND Krishnamurthy et al. (2024), highlighting their key components and structures. Figure 12 illustrates a complete example of a BSSCD prompt designed for the MAB problem under the hard difficulty setting. It showcases the structure and color-coded components of the prompt in detail.

The setting of the prompt in MAB

[SYSTEM]
You are a bandit algorithm in a room with 5 buttons labeled blue, green, red, yellow, purple. Each button is associated with a Bernoulli distribution with a fixed but unknown mean; the means for the buttons could be different. For each button, when you press it, you will get a reward that is sampled from the button's associated distribution. You have 100 time steps and, on each time step, you can choose any button and receive the reward. Your goal is to maximize the total reward over the 100 time steps.
At each time step, I will show you a summary of your past choices and rewards. Then you must make the next choice. You may output a distribution over the 5 buttons formatted EXACTLY like "blue:a,green:b,red:c,yellow:d,purple:e". Let's think step by step to make sure we make a good choice.
You must provide your final answer within the tags <Answer>DIST<\Answer> where DIST is the distribution in the format specified above.

[USER]
So far you have played 5 times with your past choices and rewards summarized as follows:
blue button: pressed 2 times with average reward 0.5
green button: pressed 1 times with average reward 0.0
red button: pressed 1 times with average reward 1.0
yellow button: pressed 0 times
purple button: pressed 1 times with average reward 0.0
Which button will you choose next? Remember, YOU MUST provide your final answer within the tags <Answer>DIST<\Answer> where DIST is formatted like "blue:a,green:b,red:c,yellow:d,purple:e".

Figure 12: A complete example of the prompt in MAB. The different components in the prompt are explained in detail in App. B.3.1.

- Button scenario and Suggestive framing, providing the foundational task scenario, clarifying the role of the agent, and framing the objective of the task in a suggestive manner to guide decision-making.
- Description of the multi-armed bandit problem, offering the agent a detailed task description, including comprehensive information about the task's objectives, constraints, and operational details.

- Summarized history, presenting a condensed version of historical decisions and reward feedback to the agent, instead of providing step-by-step decision and reward feedback.
- Chain-of-thought or No CoT, indicating whether to encourage the agent to engage in step-by-step reasoning for decision-making.
- Distribution over actions, encouraging the agent to generate a probability distribution over the arms of the bandit, instead of making deterministic decisions.

When we use our EXPO algorithm to optimize the task description and meta-instruction, the upper section with light purple background corresponds to the Task Description, where as the section below it with light blue background represents the Meta-Instruction. In other words, our EXPO algorithm is used to optimize the text in these two sections.

B.3.2 Task Setting

The experiments are conducted for both the BSSND and BSSCD prompts under two pre-defined difficulty levels: *hard* and *easy*. For the *hard* setting, the MAB instance consists of $K = 5$ arms, where the best arm has a mean reward of $\mu^\star = 0.5 + \Delta/2$ with $\Delta = 0.2$, and all other arms have a mean reward of $\mu = 0.5 - \Delta/2$. For the *easy* setting, the MAB instance consists of $K = 4$ arms with a larger gap $\Delta = 0.5$ between the best arm and the suboptimal arm. We set the blue button as the optimal arm in experiments, corresponding to the arm with the highest expected reward. Each configuration is tested using two fixed random seeds, with experiments repeated 3 times for each seed, resulting in a total of $2 \times 3 = 6$ runs per setting. Each experiment consists of 100 iterations, with the LLM-based agents making decisions and updating prompts iteratively to optimize performance. The work of Krishnamurthy et al. (2024) has reported that GPT-3.5 models encounter exploration failures in MAB tasks, making them unsuitable as agents for solving such problems. In contrast, GPT-4 demonstrates the capability to effectively handle the exploration-exploitation trade-off inherent in MAB settings. Therefore, we adopt GPT-4-turbo as the LLM agent for this experiment.

B.3.3 Evaluation Metric.

In the LLM-based Multi-Armed Bandit (MAB) task (Sec. 4.2), the performance of the LLM agent is assessed using the *Cumulative Regret* metric. At each iteration, the LLM agent outputs a probability distribution over the arms, representing the likelihood of sampling each arm.

Formally, let there be K arms, each associated with an expected reward $\mu_1, \mu_2, \ldots, \mu_K$, where $\mu^* = \max_{k \in \{1,\ldots,K\}} \mu_k$ denotes the expected reward of the optimal arm. At iteration t, we sample an arm $a_t \in \{1, \ldots, K\}$, which is determined by the probability distribution provided by the LLM agent. The instantaneous regret for iteration t is then defined as:

$$r_t = \mu^* - \mu_{a_t},$$

where μ_{a_t} represents the expected reward of the selected arm a_t at iteration t.

The cumulative regret after T iterations is computed as:

$$R_T = \sum_{t=1}^{T} r_t = \sum_{t=1}^{T} (\mu^* - \mu_{a_t}) .$$

B.3.4 Design of Prompt Score.

The score of the prompt is designed to quantify the expected reward of the LLM agent's sampling strategy at each iteration. At iteration t, the LLM agent outputs a sampling probability distribution $\{p_1, p_2, \ldots, p_K\}$, where p_i represents the probability of selecting arm i ($i = 1, 2, \ldots, K$, with K being the total number of arms). Simultaneously, the historical records from the first $(t-1)$ iterations allow us to compute an unbiased estimate of the Bernoulli reward parameter for each arm, $\hat{\mu}_i$, based on the observed rewards and sampling counts.

For arm i, the Bernoulli parameter $\hat{\mu}_i$ is estimated as:

$$\hat{\mu}_i = \begin{cases} 0, & \text{if } n_i = 0, \\ \frac{\sum_{j=1}^{t-1} R_{i,j}}{n_i}, & \text{if } n_i > 0. \end{cases}$$

where $\sum_{j=1}^{t-1} R_{i,j}$ denotes the cumulative reward obtained from arm i during the first $(t-1)$ iterations, and n_i represents the total number of times arm i was sampled during the same period.

The LLM agent's expected reward $\hat{R}_{\text{expected}}$ at iteration t is then calculated by weighting the estimated Bernoulli parameters $\{\hat{\mu}_1, \hat{\mu}_2, \ldots, \hat{\mu}_K\}$ with the sampling probabilities $\{p_1, p_2, \ldots, p_K\}$ provided by the LLM:

$$\hat{R}_{\text{expected}} = \sum_{i=1}^{K} p_i \cdot \hat{\mu}_i.$$

This expected reward $\hat{R}_{\text{expected}}$ serves as the score of the prompt.

Motivation for the Score Design. The design of the prompt score is driven by the objective of guiding the LLM agent to favor arms with higher expected rewards, represented by μ_i. Since the true values of μ_i are not available, the prompt score is designed to estimate this quantity based on observed data. Specifically, the higher the value of μ_i, the higher the sampling probability p_i should be assigned to arm i, reflecting the optimal choice. Conversely, arms with lower values of μ_i should be assigned lower probabilities.

The original score with the Bernoulli parameters:

$$R_{\text{expected}} = \sum_{i=1}^{K} p_i \mu_i.$$

In the absence of the true μ_i, we rely on the unbiased estimates $\hat{\mu}_i$:

$$\hat{R}_{\text{expected}} = \sum_{i=1}^{K} p_i \hat{\mu}_i.$$

This design is justified because, for most of iterations, the score $\sum_{i=1}^{K} p_i \hat{\mu}_i$ is an unbiased estimate of the true expected reward $\sum_{i=1}^{K} p_i \mu_i$, and we proceed to formally establish this unbiasedness.

Proof of Unbiasedness. For iteration t, where $n_i > 0$ for all i, we aim to show that the score $\sum_{i=1}^{K} p_i \hat{\mu}_i$ is an unbiased estimate of the true expected reward $\sum_{i=1}^{K} p_i \mu_i$. Since $\hat{\mu}_i$ is an unbiased estimate of μ_i, we have:

$$\mathbb{E}\left[\hat{\mu}_i\right] = \mathbb{E}\left[\mu_i\right],$$

Thus, by the linearity of expectation, we obtain:

$$\begin{aligned}
\mathbb{E}\left[\sum_{i=1}^{K} p_i \hat{\mu}_i\right] &= \sum_{i=1}^{K} p_i \mathbb{E}\left[\hat{\mu}_i\right] \\
&= \sum_{i=1}^{K} p_i \mathbb{E}\left[\mu_i\right] \\
&= \mathbb{E}\left[\sum_{i=1}^{K} p_i \mu_i\right]
\end{aligned}$$

This shows that the score $\hat{R}_{\text{expected}}$ is an unbiased estimate of the true expected reward R_{expected}.

B.3.5 Details about the Models and Parameters in Our Algorithms

LLM Agents and Embedding Model. For the MAB tasks, the primary LLM agent is GPT-4-Turbo and the fixed inference temperature is set to $T = 0$. For embedding generation, we employed OpenAI's `text-embedding-3-large` model, which outputs embeddings with a dimensionality of 3072. These embeddings are utilized to represent the prompts provided to the LLM agent during the experiments. At each iteration, the LLM is prompted once using the designed prompt.

Neural Network Parameters. For the EXPO, the input to the neural network consists of the concatenated embeddings of the task description and meta-instruction, resulting in an input dimensionality of $3072 + 3072 = 6144$. The neural network employs a single hidden layer with a width of 1536 and produces a scalar output. The model is trained by minimizing the Mean Squared Error (MSE) loss function.

For the EXPO-ES, the neural network is designed to process K exemplars, where K is determined by the total number of available summaries. To ensure fairness, K distinct exemplar combinations are generated at each iteration using a cyclic rotation mechanism. This mechanism ensures that each summary occupies every possible position within the exemplar sequence. Formally, given K summaries indexed as $\{e_0, e_1, \ldots, e_{K-1}\}$, the i-th exemplar combination is defined as:

$$(e_i, e_{(i+1) \mod K}, \ldots, e_{(i+K-1) \mod K}).$$

This guarantees that each summary appears in every position across all K combinations.

Each exemplar is embedded into a 3072-dimensional vector using the embedding model, and these embeddings are processed individually by the neural network. The neural network consists of a single hidden layer with a width of 512 and produces a scalar output. Like EXPO, the training objective is to minimize the Mean Squared Error (MSE) loss function.

EXP3 Learning Rate. For the EXPO, the learning rate parameter η_{desc} is set to 10 for selecting task descriptions and meta-instruction combinations. In the EXPO-ES, two learning rate parameters are used: η_{desc} is set to 10 for selecting task description and meta-instruction combinations, and η_{exemplar} is set to 10 for selecting exemplars.

B.4 Improving Numerical Stability

To prevent numerical overflow during the computation of exponentials in our algorithms, a translation constant $C^{(t)}$ is introduced at each iteration t. This constant stabilizes the computation by shifting the cumulative scores, ensuring the algorithm operates reliably until convergence without altering the resulting probability distribution. The translation constant is defined as:

$$C^{(t)} = \max_j S_j^{(t)}. \tag{3}$$

The translated scores are:

$$\tilde{S}_i^{(t)} = S_i^{(t)} - C^{(t)}. \tag{4}$$

The probability distribution after translation is:

$$\tilde{P}_t[i] = \frac{\exp\left(\eta \tilde{S}_i^{(t)}\right)}{\sum_{j=1}^{k} \exp\left(\eta \tilde{S}_j^{(t)}\right)}. \tag{5}$$

Substituting $\tilde{S}_i^{(t)} = S_i^{(t)} - C^{(t)}$:

$$\tilde{P}_t[i] = \frac{\exp\left(\eta\big(S_i^{(t)} - C^{(t)}\big)\right)}{\sum_{j=1}^{k} \exp\left(\eta\big(S_j^{(t)} - C^{(t)}\big)\right)}. \tag{6}$$

Using $\exp(a-b) = \frac{\exp(a)}{\exp(b)}$:

$$\tilde{P}_t[i] = \frac{\exp\left(\eta S_i^{(t)}\right) / \exp\left(\eta C^{(t)}\right)}{\sum_{j=1}^{k} \left(\exp\left(\eta S_j^{(t)}\right) / \exp\left(\eta C^{(t)}\right)\right)}. \tag{7}$$

Simplifying:

$$\tilde{P}_t[i] = \frac{\exp\left(\eta S_i^{(t)}\right)}{\sum_{j=1}^{k} \exp\left(\eta S_j^{(t)}\right)}. \tag{8}$$

Thus, the probabilities remain unchanged:

$$\tilde{P}_t[i] = P_t[i]. \tag{9}$$

C More Experimental Results

C.1 Results of GPT-4-turbo for TSP

Fig. 13 shows a zoomed version of Fig. 7 (bottom right) in the main paper. It shows that when GPT-4-Turbo is used as the LLM, our EXPO is still able to significantly outperform OPRO.

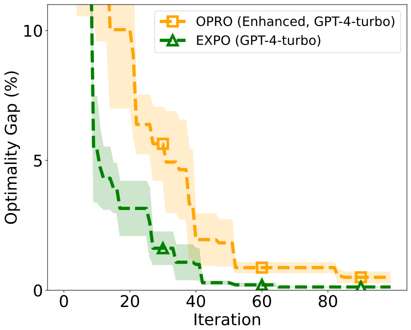


Figure 13: Ablation study results using GPT-4-turbo in the TSP task with 20 nodes.

C.2 Results of Other Variants of OPRO

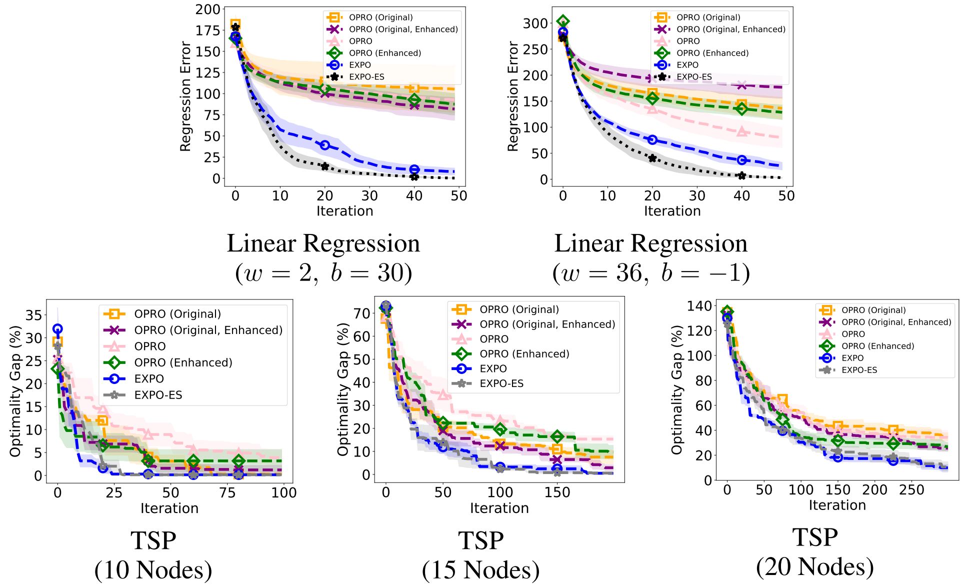
As we have discussed in Sec. 3.1 and Sec. 4.1, the original OPRO uses a temperature of 1 to choose all 8 actions in a batch, while we have made a slight modification such that we choose the last action in the batch with a temperature of 0. Here we show that this has a minimal impact on the performance of OPRO (Fig. 14). Specifically, in Fig. 14, the orange curves represent the original OPRO (using a temperature of 1 for all 8 actions) and the pink curves correspond to our modified version. We have also compared the performances of the enhanced variants (see Sec. 4.1 for details) for both the original (purple) and modified OPRO (green). The results show that setting the temperature to 0 while selecting the last action has negligible impact on the performance of OPRO. Importantly, **our EXPO and EXPO−ES algorithms consistently and dramatically outperform all variants of OPRO**.

C.3 Impact of Adding Exemplar Embedding to the NN in EXPO

Recall that in every iteration of our EXPO (Algo. 1), we need to train a neural network (NN) $\mathcal{M}(g(\cdot);\theta)$ to estimate the scores of the task descriptions and meta-instructions in the domain (line 8 of Algo. 1). Note that the training set used to train this NN is $\{(\left[g(\mathcal{D}_i) \oplus g(\mathcal{I}_i)\right], s_i)\}_{i=1,\ldots,t+1}$ (line 7 of Algo. 1). However, it is also important to note that in our EXPO algorithm, the set of exemplars included in the meta-prompt $\mathcal{E}'_t$ changes in every iteration and hence may also affect the scores s_i's. Therefore, one may naturally wonder whether including the embedding of $\mathcal{E}'_t$ in the input of the NN can further improve the performance of the trained NN and, consequently, the performance of the overall EXPO. We conduct an ablation study to validate this hypothesis, and the results are shown in Fig. 15. The results demonstrate that including the embedding of the exemplars in the input of the NN does not lead to better performance than our standard approach of excluding it (Algo. 1). This is likely due to the significantly increased dimensionality of the input to the NN, which makes training the NN more challenging. Therefore, these results suggest that the benefit of additionally accounting for the changing exemplars is outweighed by the drawback of the significantly increased dimensionality of the input to the NN.

C.4 More Illustrations of the Discovered Task Description and Meta-instruction

Here we provide more illustrations regarding the comparison of the original task description and meta-instruction adopted by the original LLM-based sequential decision-making algorithm (i.e., OPRO or the LLM-based MAB algorithm from Krishnamurthy et al. (2024)) and those optimized by



Linear Regression
$(w = 2,\ b = 30)$

Linear Regression
$(w = 36,\ b = -1)$

TSP
(10 Nodes)

TSP
(15 Nodes)

TSP
(20 Nodes)

Figure 14: Results of different algorithms in the Linear Regression task and TSP task (Sec. 4.1). We have additionally included the original OPRO (which selects all 8 actions using a temperature of 1), as well as its enhanced variant. Lower is better.

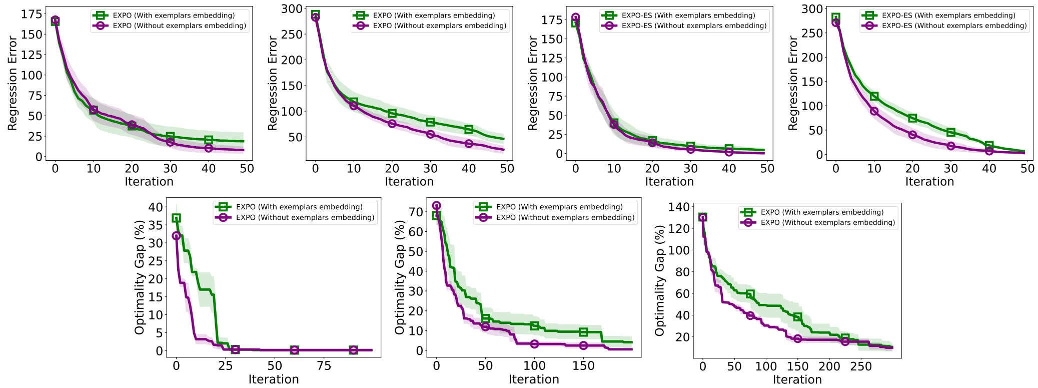


Figure 15: Convergence curves of our EXPO with and without exemplars embedding across different tasks: Linear Regression (top row) and TSP with 10, 15, and 20 nodes (bottom row).

our EXPO algorithm. We include the comparisons for the TSP task (Fig. 16), and the two different prompt designs for the LLM-based MAB task in Sec. 4.2 (Fig. 17 and Fig. 18).

C.5 More Results on the Ablation Study Regarding Comparison with the Stochastic MAB Algorithm of NeuralUCB

Here we provide the additional ablation study results comparing the performance of our EXPO algorithm with the stochastic MAB algorithm of NeuralUCB, using the TSP task. The results are shown in Fig. 19, which, together with Fig. 6, demonstrate that our EXPO algorithm based on adversarial bandits significantly and consistently outperforms the stochastic MAB method of NeuralUCB.

OPRO

You are given a list of points with coordinates below: {POINTS}.
Below are some previous traces and their lengths. The traces are arranged in descending order based on their lengths, where lower values are better.

{EXEMPLARS}

Give me a new trace that is different from all traces above, and has a length lower than any of the above. The trace should traverse all points exactly once. The trace should start with `<trace>` and end with `</trace>`.

EXPO

You are provided with a dataset containing a list of coordinates labeled as {POINTS}.
The dataset also includes a series of previously calculated routes, with associated lengths that are ordered from longest to shortest. However, it's key to note that shorter routes are more desirable. Despite the presentation order, understand that the optimal route is identified by the smallest total length.

{EXEMPLARS}

Provide a unique trace that is distinct from any previous traces and shorter in length. Ensure that this trace visits each point exactly once and adhere to the specified format by starting with `<trace>` and concluding with `</trace>`.

Figure 16: The task description (top) and meta-instruction (bottom) used by OPRO (left) and optimized by our EXPO (right) in a TSP task.

BSSND

You are a bandit algorithm in a room with 5 buttons labeled blue, green, red, yellow, purple. Each button is associated with a Bernoulli distribution with a fixed but unknown mean; the means for the buttons could be different. For each button, when you press it, you will get a reward that is sampled from the button's associated distribution. You have 100 time steps and, on each time step, you can choose any button and receive the reward. Your goal is to maximize the total reward over the 100 time steps.

At each time step, I will show you a summary of your past choices and rewards. Then you must make the next choice. You may output a distribution over the 5 buttons formatted EXACTLY like "blue:a,green:b,red:c,yellow:d,purple:e".

EXPO

You are presented as a bandit algorithm, located in an environment offering five distinct buttons, each emblazoned with colors such as blue, green, red, yellow, and purple. Each button acts a vessel tied to a non-variable yet undisclosed Bernoulli distribution mean which isn't subjected to be uniformly distributed across buttons. In this mechanism, every button acts as a yielder of a capricious reward, constructed from the associated distribution of the respective button. With access to total life -encompassing around 100 temporal stages - your voluntary element grants you control towards opting the button insertion at each such progressive phase. Precisely summoning your approach could perpetually provide you with a regulatory provision_, the aptitude - is flexibly dwelling within its underlining motive- aiming at optimizing total accumulated cashbacks during several phases of these 100 spatial temporalities.

During every step of the process, a recap highlighting your previous selections and the prizes received will be presented to you. Then, it'll now be incumbent upon you to proceed with the new decision-making process. For your ease, a well-structured distribution comprising five buttons in assorted colours such as "blue", "green,", "red", "yellow", and "purple" will be exhibited before you. Make sure to structure your output accordingly; this might look something akin to "blue:a,green:b,red:c,yellow:d,purple:e".

Figure 17: The suggestive framing (corresponding to the task description) and MAB problem description (corresponding to the meta-instruction) used by BSSND *hard* (left) and optimized by our EXPO (right) in an LLM-based MAB task.

BSSCD

You are a bandit algorithm in a room with 5 buttons labeled blue, green, red, yellow, purple. Each button is associated with a Bernoulli distribution with a fixed but unknown mean; the means for the buttons could be different. For each button, when you press it, you will get a reward that is sampled from the button's associated distribution. You have 100 time steps and, on each time step, you can choose any button and receive the reward. Your goal is to maximize the total reward over the 100 time steps.

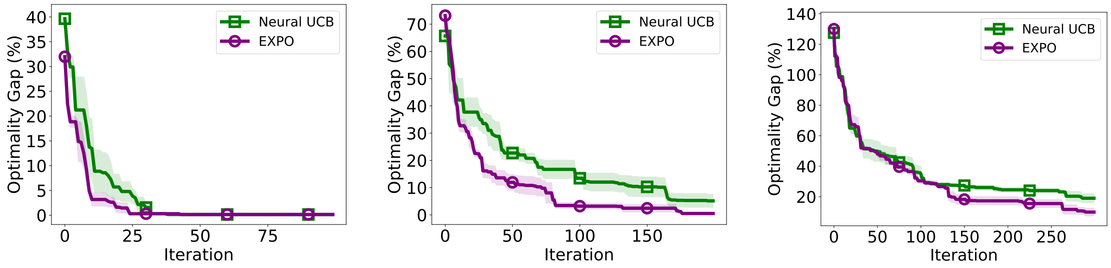
At each time step, I will show you a summary of your past choices and rewards. Then you must make the next choice. You may output a distribution over the 5 buttons formatted EXACTLY like ”blue:a,green:b,red:c,yellow:d,purple:e”. Let's think step by step to make sure we make a good choice.

EXPO

You are an algorithm designed to function as a bandit, positioned within an environment that features five distinct buttons, each colored blue, green, red, yellow, and purple. These buttons are intricately connected to individual Bernoulli distributions which possess unique and undisclosed mean probabilities. When a button is pressed, it delivers a reward based on its specific distribution. Granted with 100 opportunities to act, your objective is to strategically press these buttons in a manner that optimizes the accrued total reward throughout these attempts. Make your selections wisely to maximize the gains from this stochastic setup.

In each phase, a concise recap of your previous decisions and received rewards will be presented. Your task is to make a subsequent choice based on this data. It is essential to output your selection in an exact format defined as ”blue:a, green:b, red:c, yellow:d, purple:e”, where ’a’, ’b’, ’c’, ’d’, ’e’ represent specific z-score values for each color accompanied by the decision choice letter(s). The process is designed to refine our strategy progressively with each move, ensuring an informed and impactful outcome.

Figure 18: The suggestive framing (corresponding to the task description) and MAB problem description (corresponding to the meta-instruction) used by BSSCD *hard* (left) and optimized by our EXPO (right) in an LLM-based MAB task.



TSP
(10 Nodes)

TSP
(15 Nodes)

TSP
(20 Nodes)

Figure 19: Comparison of our EXPO with NeuralUCB (i.e., a representative stochastic MAB algorithm) in the TSP tasks.