Plum: Prompt Learning using Metaheuristics

Rui Pan[1*], **Shuo Xing**[2*], **Shizhe Diao**[1*], **Wenhe Sun**[1*†], **Xiang Liu**[1†], **Kashun Shum**[1], **Jipeng Zhang**[1], **Renjie Pi**[1], **Tong Zhang**[3]

[1]The Hong Kong University of Science and Technology [2]Texas A&M University
[3]University of Illinois Urbana-Champaign
{rpan,sdiaoaa,ksshumab,jzhanggr,rpi}@ust.hk, shuoxing@tamu.edu,
sunwenhe@mail.nankai.edu.cn, xliu886@connect.hkust-gz.edu.cn,
tongzhang@tongzhang-ml.org

Abstract

Since the emergence of large language models, prompt learning has become a popular method for optimizing and customizing these models. Special prompts, such as Chain-of-Thought, have even revealed previously unknown reasoning capabilities within these models. However, the progress of discovering effective prompts has been slow, driving a desire for general prompt optimization methods. Unfortunately, few existing prompt learning methods satisfy the criteria of being truly "general", i.e., automatic, discrete, black-box, gradient-free, and interpretable all at once. In this paper, we introduce metaheuristics, a branch of discrete non-convex optimization methods with over 100 options, as a promising approach to prompt learning. Within our paradigm, we test six typical methods: hill climbing, simulated annealing, genetic algorithms with/without crossover, tabu search, and harmony search, demonstrating their effectiveness in white-box and black-box prompt learning. Furthermore, we show that these methods can be used to discover more human-understandable prompts that were previously unknown in both reasoning and image generation tasks, opening the door to a cornucopia of possibilities in prompt optimization.

1 Introduction

The advent of powerful large language models (LLMs) (Devlin et al., 2018; Radford et al., 2019; Brown et al., 2020; Team et al., 2023; Roziere et al., 2023; Touvron et al., 2023a,b) has paved the way for numerous real-world applications, including multi-round chat (OpenAI, 2023; Touvron et al., 2023b), instruction following (Ouyang et al., 2022) and reasoning (Roziere et al., 2023; Azerbayev et al., 2023). To further enhance the performance of LLMs in specific domains, additional optimization techniques are commonly employed, such as fine-tuning (Chiang et al., 2023), adapter methods (Hu et al., 2021; Dettmers et al., 2023), alignment approaches (Ouyang et al., 2022; Dong et al., 2023), and prompt learning (Sun et al., 2022b; Diao et al., 2022). Notably, prompt learning distinguishes itself from other methods by eliminating the need for gradient information from models, resulting in substantially reduced memory consumption and computational resource requirements. Furthermore, prompt learning often yields interpretable outcomes, which help researchers and engineers intuitively understand its effectiveness, thereby being beneficial in inspiring more generalizable prompts for various tasks (Prasad et al., 2022; Guo et al., 2023; Yu et al., 2023).

Since the introduction of prompt engineering and prompt learning, significant advancements have been made in the discovery of effective prompts. A noteworthy illustration is Chain-of-Thought (COT), whereby the simple inclusion of accurate deduction steps for few-shot examples within the original prompt empowers LLMs to achieve substantial performance improvements in reasoning tasks (Wei et al., 2022b). An even more impressive result is Zero-shot-COT (Kojima et al., 2022), where adding the magic phrase “Let’s think step by step” produces a remarkable accuracy gain of over 10% across multiple models engaged in a diverse spectrum of reasoning tasks.

However, the quest for such highly effective prompts remains unfulfilled, underscoring the need for tools that accelerate the discovery process. Ideally, these prompt learning tools should possess a notable level of generality, while simultaneously meeting the following criteria:

- Automatic: since human involvements are normally expensive and time-consuming.
- Discrete: as commercial LLMs usually provide APIs that only accept discrete in-

[*] Equal contribution.
[†] Work done during internship at the Hong Kong University of Science and Technology.

Findings of the Association for Computational Linguistics: ACL 2024, pages 2177–2197
August 11-16, 2024

puts (OpenAI, 2023; Team et al., 2023).

- Black-box: applicable to black-box LLMs.
- Gradient-free: hence still possess the good property of low memory cost and time cost.
- Interpretable: thus researchers can understand the logics behind its effectiveness and extend its applications with ease.

Regretfully, to the best of our knowledge, only a limited number of prompt learning methods meet the criteria for being considered "general" as per the aforementioned definition. Manual prompt engineering techniques employing hand-crafted prompts (Mishra et al., 2021; Kojima et al., 2022; Bsharat et al., 2023) lack automation, while continuous prompt learning approaches (Liu et al., 2021a; Li and Liang, 2021) fail to satisfy the requirements of discreteness and interpretability. Prompt learning methods employing reinforcement learning (Deng et al., 2022) often necessitate an additional neural network, the learning process of which is typically non-interpretable, and their theoretical guarantees are only attainable under stringent assumptions.

In this paper, we present a novel approach to simultaneously achieve these desirable properties by integrating prompt learning with metaheuristics. By treating prompt learning as a non-convex discrete optimization problem within a black-box framework, we harness the potential of metaheuristics, which offer interpretable and automated optimization processes. Moreover, this domain of optimization research encompasses a vast array of over 100 distinct algorithms (Hussain et al., 2019), ushering in a new paradigm for prompt learning. Many of these algorithms even boast robust theoretical guarantees, ensuring the discovery of both local and even global optimal solutions within limited time frames (Locatelli, 2000; Glover and Hanafi, 2002; Schmitt, 2004; Dorigo et al., 2006; He et al., 2018).

2 Related Work

2.1 Prompt Learning

Prompt-based learning is an effective approach that harnesses the power of large language models (LLMs) to facilitate downstream tasks by extracting relevant knowledge. This approach offers several advantages, notably in terms of computational efficiency, as only a small set of parameters needs to be optimized, in contrast to traditional methods that require fine-tuning the entire LLM for each task. Prompting methods can be categorized into discrete prompts (Davison et al., 2019; Wallace et al., 2019; Jiang et al., 2020; Shin et al., 2020; Haviv et al., 2021; Yuan et al., 2021; Gao et al., 2021; Ben-David et al., 2022; Su et al., 2022a; Diao et al., 2022; Datta et al., 2023; Pan et al., 2024) and continuous prompts (Hambardzumyan et al., 2021; Zhong et al., 2021; Han et al., 2021; Li and Liang, 2021; Qin and Eisner, 2021; Liu et al., 2021a) based on the format of prompts. Discrete prompts are typically represented as token sequences or natural language phrases, while continuous prompts are designed as sequences of vectors. Significant advancements have been made in leveraging LLMs for reasoning tasks using chain-of-thought prompting techniques. Notable progress is evidenced by the research of Wei et al. (2022a); Wang et al. (2022); Zhou et al. (2023); Zhang et al. (2022); Shum et al. (2023); Diao et al. (2023), in which well-structured prompts guide the models to showcase superior reasoning capabilities, surpassing previous benchmarks. Further studies in prompt attacks (Yu et al., 2023; Zhu et al., 2023) have also utilized several of these advancements to improve the robustness of modern LLMs. However, most of the prior studies on prompting have been limited to a white-box setting, requiring access to all parameters of a large language model. The reliance on white-box optimization methods has become less practical in light of the widespread use of closed-source black-box models.

2.2 Black-Box Prompt Learning

Black-box prompt learning (BPL) (Diao et al., 2022; Sun et al., 2022b,a) represents a promising research direction that tunes prompts without requiring access to the parameters and gradients of the underlying LLMs. BPL proves particularly valuable for closed-source models, which have demonstrated superior performance compared to open-source models (Liang et al., 2022). This advancement allows for the effective optimization of closed-source models, overcoming the previous limitations posed by the unavailability of model parameters. Similar to white-box prompts, black-box prompts can be categorized into discrete prompts (Diao et al., 2022; Prasad et al., 2022; Deng et al., 2022; Hou et al., 2022; Cheng et al., 2023) and continuous prompts (Sun et al., 2022b,a; Su et al., 2022b). However, in the black-box setting, discrete prompts prove more practical as the interface of LLMs only accepts discrete inputs and cannot process continuous vectors.

2.3 Metaheuristics

There have been efforts combining the power of metaheuristics with black-box prompt learning in the past literature. For instance, GrIPS (Prasad et al., 2022) applied greedy search methods, as well as simulated annealing (Kirkpatrick et al., 1983), to search better prompts with simple edit operations such as addition, swap, paraphrase, and deletion. GPS (Xu et al., 2022) utilized genetic algorithms for few-shot prompt learning, improving handcrafted prompts on multiple tasks. Additionally, Kumar and Talukdar (2021) and Lu et al. (2021) investigated the impact of example orders in few-shot learning settings, respectively improving performance through the use of genetic algorithms and alternate search heuristics. However, these methods are limited in their applicability, focusing on specific prompt learning settings and failing to fully explore the inherent potential of discrete optimization, which is a bridge between black-box prompt learning and general metaheuristics. As an attempt to overcome this limitation, (Guo et al., 2023) investigated the performance of combining the rephrasing power of LLMs with Genetic Algorithm and Differential Evolution, showing superior performance over hand-crafted prompts. However, compared with our work, Guo et al. (2023) focused more on leveraging the paraphrasing ability of LLMs and restricted search strategy to only evolutionary algorithms, which is a subset of metaheuristics.

Metaheuristics is a well-established and versatile branch of discrete optimization, known for its effectiveness in solving various non-convex optimization problems. This field has witnessed the development of over 100 different methods (Hussain et al., 2019), which have been successfully applied to a wide range of problem domains. For instance, they have been utilized in Neural Architecture Search (Elsken et al., 2019; Liu et al., 2021b), Travelling Salesman Problem (Ouaarab et al., 2014; Hussain et al., 2017; Grabusts et al., 2019), and Program Search (Chen et al., 2023). Prominent algorithms in this field include simulated annealing (Kirkpatrick et al., 1983), genetic algorithms (Holland, 1992), tabu search (Glover, 1986), harmony search (Geem et al., 2001), ant colony optimization (Dorigo and Gambardella, 1997), among others. However, the integration of these powerful metaheuristics with black-box prompt learning remains largely unexplored, presenting rich opportunities for further research in this direction.

3 Methods

In this paper, we propose Prompt learning using metaheuristic (`Plum`), a general paradigm that enables the application of numerous metaheuristics in the setting of discrete black-box prompt learning. Based on this framework, the flexible composition of different techniques becomes possible, leading to a "template" for designing general prompt learning algorithms, guaranteed to simultaneously satisfy the properties of automatic, discrete, black-box, gradient-free, and interpretable.

3.1 Plum in General

Algorithm 1 `Plum` in General

Required:
(1) A well-defined neighbor set $\mathcal{N}(\mathbf{p})$ for $\forall \mathbf{p} \in \Omega$;
(2) a metaheuristics algorithm $\mathcal{A}$;
(3) metaheuristics-dependent hyperparameters θ and functions $\mathcal{F}$.

Input: An initialized prompt $\mathbf{p}_0$ and an objective $f(\mathbf{p})$ to be maximized (or minimized).

1: $\mathbf{p}_* \leftarrow \mathcal{A}_{\mathcal{N},\theta,\mathcal{F},f}(\mathbf{p}_0, \mathcal{M})$
2: **return** $\mathbf{p}_*$

The framework of `Plum` is outlined in Algorithm 1, which encompasses four fundamental and orthogonal elements. The first element is a well-defined neighborhood $\mathcal{N}(\cdot)$ in the discrete prompt search space. In practice, any operations that transform a prompt into another prompt can be used to define such neighborhoods. For instance, edit operations in GrIPS (Prasad et al., 2022) can convert a prompt $\mathbf{p} = [\text{Let}, \text{us}, \text{think}]$ to a new prompt $\mathbf{p}' = [\text{Let}, \text{us}, \text{brainstorm}]$ via paraphrase, and all such possible conversions form prompt $\mathbf{p}$'s neighborhood

$$\mathcal{N}(\mathbf{p}) = \{\mathbf{p}' | \exists e \in \mathcal{E}, s.t.\ \mathbf{p}' = \texttt{edit}(\mathbf{p}, e)\}.$$

This abstraction of neighborhood decouples prompt transformations and search algorithms, making the flexible combination of those techniques possible.

The second part is the core metaheuristics $\mathcal{A}$, such as simulated annealing (Kirkpatrick et al., 1983), tabu search (Glover, 1986), genetic algorithms (Holland, 1992). Any discrete optimization metaheuristics can fit into this part.

Neighborhood $\mathcal{N}(\mathbf{p})$	**Metaheuristics** $\mathcal{A}$	**Hyperparameter** θ	$\mathcal{F}$	Name
Candidates after editing (Prasad et al., 2022)	Hill Climbing	-	-	`Plum-HC`
	Simulated Annealing	-	Temperature schedule $T(i)$	`Plum-SA`
	Genetic Algorithms (mutation only)	k tournament selections	-	`Plum-GA-M`
	Genetic Algorithms (with crossover)	1) k tournament selections; 2) p_{mutation}: mutation rate	Crossover function: $\langle \mathbf{p}_1, \mathbf{p}_2 \rangle \to \mathbf{p}$	`Plum-GA-C`
	Tabu Search	N_{tabu} slots in Tabu list	Tabu function: $\langle \mathcal{T}, \mathbf{p} \rangle \to \{0, 1\}$	`Plum-TS`
	Harmony Search	1) N_H harmony search memory; 2) k_s number of segments; 3) $HMCR \in [0, 1]$: harmony memory considering rate; 4) $PAR \in [0, 1]$: pitching adjust rate	-	`Plum-HS`

Table 1: Implemented `Plum` algorithms.

Notably, various metaheuristics come with their own specific hyperparameters and functions that need to be considered within the `Plum` paradigm. For instance, in the case of simulated annealing, a temperature scheduler is required for regulating the annealing speed, where an appropriate schedule can greatly improve the search efficiency. Similarly, general genetic algorithms incorporate a "crossover" operation, which swaps segments of two prompts and serves as a global search mechanism to expand its search scope.

During the optimization process, the objective is to maximize (or minimize) a provided function $f(\mathbf{p})$, which typically corresponds to task-specific performance metrics of the target LLM. For instance, under the setting of GPT-3-babbage (Brown et al., 2020) on Commonsense Question Answering (Talmor et al., 2019), the function $f(\mathbf{p})$ represents the final accuracy achieved by augmenting GPT-3-babbage with the prompt $\mathbf{p}$. The primary goal of the metaheuristics is to search for the optimum $\mathbf{p}$ which maximizes this accuracy.

By clearly stating those key components in `Plum`, we are now prepared to instantiate `Plum` with specific metaheuristics and neighborhood definitions that can perform real-world prompt learning tasks.

3.2 Plum in Practice

As a proof of concept, we realize `Plum` with several popular metaheuristics, as listed in Table 1. Detailed pseudocodes of those methods are available in Appendix C.

All those algorithms are successful products of metaheuristics under the setting of prompt learning. By combining the edit operations employed in GrIPS (Prasad et al., 2022) with simulated annealing (Kirkpatrick et al., 1983), we developed an algorithm capable of recovering global optimum with proper temperature scheduler (Granville et al., 1994). In this context, we define the global optimum as the prompt that maximizes the objective function $f(\mathbf{p})$ within the set of reachable prompts starting from the initial prompt $\mathbf{p}_0$.

Furthermore, replacing simulated annealing with genetic algorithms gives birth to `Plum-GA-M` and `Plum-GA-C`, whose metaheuristics prototypes are proven to converge to global optimum given specific mutation and crossover operations (Schmitt, 2004). Compared with `Plum-GA-M`, `Plum-GA-C` adopts an extra operation of crossover, which enables more aggressive searches. Intuitively, `Plum-GA-C` > `Plum-GA-M` > `Plum-SA` > `Plum-HC` in terms of their exploration power, which is at the expense of increased search steps and the number of evaluations of the objective function $f(\mathbf{p})$.

Similarly, adopting Tabu Search (Glover, 1986) and Harmony Search (Geem et al., 2001) brings forth `Plum-TS` and `Plum-HS`, both are API-efficient methods according to our experiments. Here Tabu Search introduces a Tabu list to prohibit certain candidates from being revisited, hence avoids the search process from getting stuck in a local optimum. Meanwhile, Harmony Search obtains its inspiration from musical composition, utilizing a harmony search memory with past prompts and generates the new prompt by combining segments

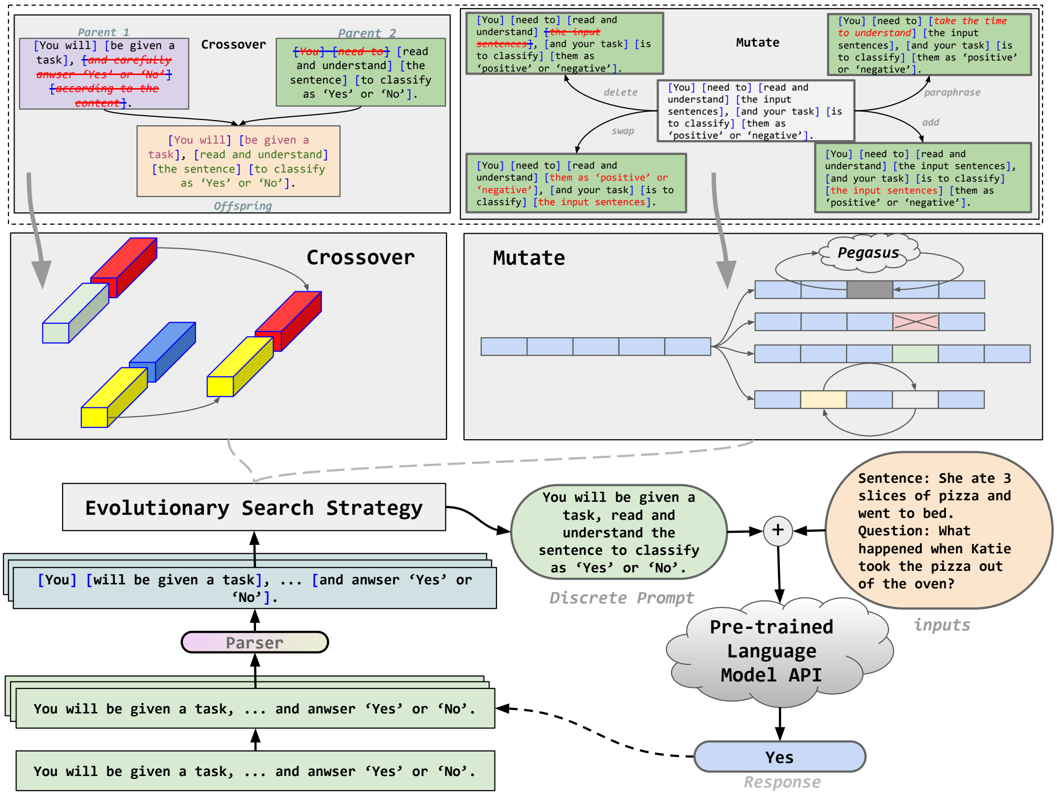


Figure 1: Illustration of Plum-GA-C and its possible extension with LLMs.

from them.

3.3 Plum-GA-C

Here we illustrate one specific algorithm Plum-GA-C as an example of instantiating a Plum algorithm. Plum-GA-C contains three stages: parsing, editing, and scoring, which are described as follows.

3.3.1 Stage 1: Parsing

Initially, we parse the initial prompt I into a sequence of phrases using a CRF-based constituency parser (Zhang et al., 2020a). Subsequently, we iteratively merge the leaf nodes of the obtained constituency tree until several disjoint phrase-level constituents are generated.

3.3.2 Stage 2: Editing

Mutation Mutation is a combination of edit operations inside the prompt. Following Prasad et al. (2022), we introduce four mutation operations including: `delete`, `add`, `swap`, and `paraphrase`. `delete` randomly drops some phrases while `addition` adds some randomly sampled phrases back at a random position. `swap` swaps two phrases, and `paraphrase` replaces a phrase with a new phrase generated by a paraphrasing model. We apply PEGASUS (Zhang et al., 2020b) as the paraphrase model to obtain new phrases. The upper right corner of Figure 1 provides an illustrative depiction of the mutation process.

Crossover Crossover is a combination of edit operations across two prompts. To perform crossover, we divide each prompt into two sub-sequences at a randomly selected position. From each prompt, we randomly choose one sub-sequence and these selected sub-sequences are then combined to create a new sequence as the prompt. The upper left corner of Figure 1 provides a visual representation of the crossover process.

3.3.3 Stage 3: Scoring

The optimization process relies on performance signals acquired through the scoring process. Initially, we construct a score set $S = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ by randomly sampling n examples from the training split. Subsequently, a score function is computed based on the accuracy of predictions.

3.4 Plum-HS

We also introduce Plum-HS, a novel algorithm that integrates Harmony Search into prompt learning.

Notably, it marks the first instance of Harmony Search being successfully applied to prompt optimization tasks. The process is quite similar to composing new songs, where new prompts are generated by combining segments from past prompts together. The procedure starts with a base prompt pending to be optimized, which is first divided into segments of words, and inserted into the harmony memory.

At each iteration, a new prompt is generated by composing k_s segments from prompts in the harmony search memory, where the j-th segment is the exact copy of the j-th segment from a randomly sampled prompt in the memory. After this composition procedure, a pitch finetuning process is then applied, which edits the segment with a small probability in a similar manner as the mutation operation in Section 3.3.2. The only difference is that it utilizes two kinds of pitch change operations: small pitch changes which only do paraphrasing, and big pitch changes which include all editing operators. The generated new prompt is then evaluated and inserted back into the harmony search memory, where the memory only retains N_H prompts with top scores.

Intuitively speaking, `Plum-HS` has a similar exploration power as `Plum-GA-C` and adopts a more flexible manner to replace long segments in the middle of the prompt. According to our experiments, it is also API-efficient, which achieves better performance while consuming significantly fewer API calls compared to other methods.

Besides `Plum-GA-C` and `Plum-HS`, details of other algorithms are also available in Appendix C.

4 Experiments

We conduct three types of experiments to demonstrate the empirical superiority of proposed algorithms. As a starting point, we first show `Plum`'s computational efficiency by applying them to general prompt learning tasks with white-box models, where `Plum` produces nontrivial improvements over baselines within a fixed time limit. We further investigate `Plum` on prompt learning tasks with black-box models, where only discrete-input APIs are available. On these tasks, `Plum` also obtains significant performance gains with much fewer API calls. Eventually, we show that `Plum` is capable of discovering effective prompt patterns in both Chain-of-Thought and text-to-image generation tasks, making it a promising paradigm for boosting research in prompt learning.

Methods	Accuracy(%)
BBT (Sun et al., 2022b)	53.67±1.71[‡]
BDPL (Diao et al., 2022)	53.13±0.61
GrIPS (Prasad et al., 2022)	53.67±0.87
APO (Pryzant et al., 2023)	54.63±0.37
`Plum-HC`	53.83±0.46
`Plum-SA`	53.92±0.41
`Plum-GA-M`	53.21±0.60
`Plum-GA-C`	54.38±0.47
`Plum-TS`	54.18±0.05
`Plum-HS`	**55.04±0.56**

Table 2: Impact of different search strategies with task-agnostic instruction and Instruction-Only prompts with gpt2-large as backbones. Here we set the time limit to 45 minutes for all methods.

[‡] We incorporate error bars for all reported results in this paper to illustrate the variability, where $\pm$ represents a plus or minus the standard deviation.

4.1 Plum for White-Box Prompt Learning

Dataset Our experiments are conducted on a subset of the Natural-Instructions datasets v2.6 release (Mishra et al., 2022), specifically focusing on eight binary classification tasks (task 019, task 021, task 022, task 050, task 069, task 137, task 139, and task 195). This dataset evaluates the instruction following ability of models, where proper prompt design plays an important role.

Baselines We compare our method with three popular prompt learning methods:

- **BDPL** (Diao et al., 2022): using variance-reduced policy gradient algorithm to estimate the gradients of parameters in the categorical distribution of each discrete prompt.
- **BBT** (Sun et al., 2022b): optimizing continuous prompts by covariance matrix adaptation evolution strategy.
- **GrIPS** (Prasad et al., 2022): generating discrete prompts by performing phrase-level operations, and selecting the best ones.
- **APO** (Pryzant et al., 2023): automatically improve the initial prompt using beam search guided by textual “gradient”.

Experiment Setup To make the comparison fair for all algorithms, we follow the same editing operators as GrIPS (Prasad et al., 2022), which generally generates discrete prompts by performing phrase-level operations and selecting the best ones. We utilize GPT2-large (Radford et al., 2019) for

Backbone	Methods	Max Iteration	Batch Size	Accuracy(%)	API Calls
GPT3-babbage (without API Calls limit)	BDPL (Diao et al., 2022)	50	32	57.38	10000
	GrIPS (Prasad et al., 2022)	10	4	54.41±0.55	7740
	`Plum-HC`	50	20	56.25±0.27	10588
	`Plum-SA`	50	20	54.92±0.78	13900
	`Plum-GA-M`	50	20	56.04±0.97	10098
	`Plum-GA-C`	50	20	56.63±0.97	13367
	`Plum-TS`	50	20	54.63±0.83	1752
	`Plum-HS`	50	20	**59.63±0.80**	5494
GPT3-babbage (with API Calls limit 8000)	BDPL	40	32	56.25	–
	GrIPS	10	4	54.41±0.55	–
	`Plum-HC`	50	20	55.21±0.48	–
	`Plum-SA`	50	20	56.87±1.62	–
	`Plum-GA-M`	50	20	55.63±0.47	–
	`Plum-GA-C`	50	20	57.75±0.74	–
	`Plum-TS`	50	20	54.92±1.26	–
	`Plum-HS`	50	20	**59.75±0.81**	–

Table 3: Impact of different search strategies with task-agnostic instruction and Instruction-Only prompts with GPT3 as backbone.

the backbone model of prompt learning. To accommodate computational resource limitations, we set the batch size to 1 and the time-out threshold of the searching process to 45 minutes for all methods. All methods are given a task-agnostic instruction: *You will be given a task. Read and understand the task carefully, and appropriately answer [list_of_labels].* Here *[list_of_labels]* serves as a placeholder to be replaced by an actual list of labels.

Results As demonstrated in Table 2, with limited time resources, `Plum` can provide generally on-par or better performance than baseline prompt learning methods. It is worth noticing that this is achieved even without the need for model gradients or any assistance from external LLMs, which shows the potential of `Plum` for offline prompt optimization scenarios where efficiency is of great importance. More experimental results are available in Appendix B.1.

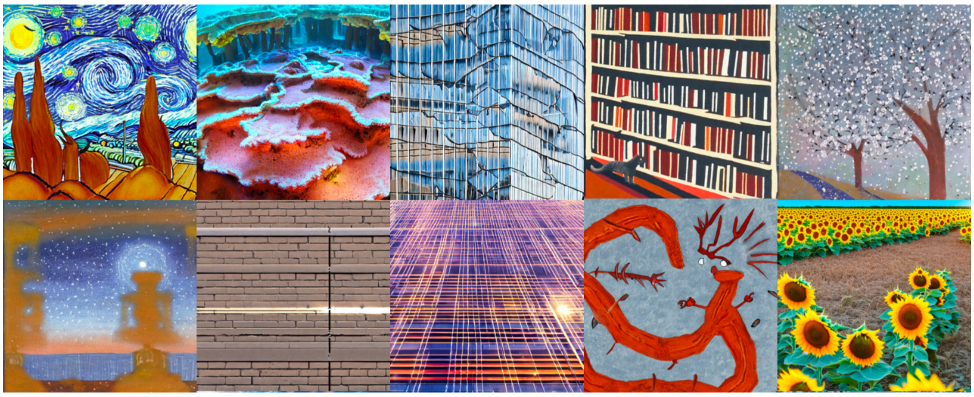
4.2 Plum for Black-Box Prompt Learning

Experimental Setup The experimental setup follows the same setting as white-box prompt learning, except for the backbone model, where we adopt GPT3-babbage instead. Notice that BBT (Sun et al., 2022b) is no longer applicable in this setting since it requires continuous gradients in its prompt learning processes.

Results Table 3 lists the performance of `Plum`-series algorithms, where under both scenarios of a limited number of iterations and API calls, `Plum-HS` achieves the best performance. Specifically, `Plum-HS` obtains this superior performance at a cost of much fewer API calls, rendering it a promising candidate for other prompt-tuning tasks. It is also worth noticing that given approximately the same amount of API calls, all `Plum` algorithms surpass GrIPS by a non-trivial margin.

4.3 Plum for Prompt Discovery

Text-to-Image Generation As demonstrated by images in Table 4, `Plum` can also be applied to text-to-image generation tasks and obtain non-trivial improvements even without the assistance of external LLMs like GPT-3 (Brown et al., 2020). By utilizing an offline image evaluator called PickScore (Kirstain et al., 2023), optimizing image relevance to the original prompt can be formulated as a black-box prompt learning problem, where a prompt's score is defined as the averaged PickScore of the images generated from the prompt. PickScore is a CLIP-based scoring function trained on a corpus of high-quality image text pair dataset. This makes improving the image quality quantitatively possible. For example, in the fourth image of Table 4, where the image topic is "cat in a library", `Plum-HS` augments the original prompt "*A cat prowling in a library at night, books and shadows, silent observer*", and replaces it with a high-score prompt "*A Is that true? Not at all cat prowling in shadows And books shadows*". Notice the resultant prompt highlights the key information "cat, books, shadows" and appends a prefix "Is that true?" with a sense of mystery. To the best of our



(a) SD-2.1 (768 × 768 resolution, Average PickScore = 50.0)

(b) `Plum-HS` augmented SD-2.1 (768 × 768 resolution, Average PickScore = 80.6)

Table 4: `Plum-HS` augmented prompts for `stable-diffusion-2-1`. Image topics: **Top**: 1) Starry night, 2) An underwater city, 3) Reflection, 4) Cat in a library, 5) Moonlight bathes blossom; **Bottom**: 1) Moonlight and city lights, 2) Boundary, 3) Sun glare on a skyscraper, 4) Fiery beast, 5) Sunflower.

knowledge, this pattern was unexplored before and can be beneficial for future prompt engineering development in this field. More experimental results and details are available in Appendix A.

Reasoning Another important feature of `Plum` is its compatibility with external LLM search operators for paraphrasing, which allows the discovery of interpretable prompt patterns. We apply this version of `Plum` to improve task-oriented LLMs' performance, where the detailed cases are presented in Table 6 of Appendix A. Via studying the characteristics of the searched prompts, we found that the performance of initial prompts can be boosted by further completion of the logical chain. This way, the logical coherence of prompts can be improved while still retaining their intended meaning. For instance, *'If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot (total number of cars in the parking lot + 2)'*, where GPT-3 completes the content inside the parentheses, leading to an easier Chain-of-Thought reasoning for LLMs.

Furthermore, it is observed that the quality of the prompts can be enhanced by incorporating a more comprehensive and lucid explanation of the statement. To illustrate, in a case of prompting for AQuA, GPT3.5 provides an initial explanation of the term *'keystrokes'* with *'(the act of pressing a key on a keyboard)'* to ensure clarity, and further gives a precise description for the range *'from 1 to 500'* with *'(inclusive)'*. This suggests that knowledge-augmented prompt learning methods can be beneficial for black-box prompt tuning.

5 Discussion

`Plum` with LLM-driven Mutators LLM-based prompt tuning methods are emerging as one of the most competitive approaches in prompt learning. Recent works like Pryzant et al. (2023); Guo et al. (2023) have shown significant performance improvements through utilizing the LLM-driven prompt mutators, highlighting the success of such approaches. Compared with them, `Plum` offers a complementary and orthogonal paradigm. For example, by incorporating the LLM-based operators in Plum algorithms, our Chain-of-Thought task experiments in Appendix B.2 demonstrate effectiveness in generating high-quality, interpretable few-shot-CoT prompts for reasoning tasks (as shown in Table 6). This suggests that LLMs can be a powerful tool for prompt exploration within the Plum paradigm, particularly when computational resources are abundant.

Strengths of Different Plum Algorithms In both white-box and black-box settings, our experiments demonstrate that Plum-HS achieves significant performance gains on prompt tuning tasks against baselines and the other Plum algorithms. This is because it takes into account the composability of different prompt segments, which can be seen as an extension of evolutionary algorithms. Notably, in tasks like image generation (detailed in Section 4.3), we observe that updates to different segments can occur in parallel, highlighting this approach's efficiency.

On the other hand, while two baseline methods may have superior performance over most other Plum algorithms, those metaheuristics possess nice properties that can be beneficial for the community.

- Plum-HC stands out for its simplicity and efficiency among Plum family. It shares the same form of GrIPS (Prasad et al., 2022) and is one of the most commonly adopted paradigms in LLM-based methods. Plum-HC can serve as a foundational algorithm for developing new mutators. Its straightforward nature allows for easier debugging and comparison when developing more complex algorithms in the Plum family.

- For Plum-SA, it boasts a quite simple form, which only differs one line from Plum-HC. Crucially, it guarantees to find the global optimum with the infinite time horizon. This characteristic makes it valuable for future theoretical analyses of discrete prompt learning.

- Plum-GA-M and Plum-GA-C, the genetic algorithm, can be considered one of the most widely adopted algorithms in evolutionary search, whose results can be useful for future comparisons on the same tasks for other researchers. Additionally, our experiments also demonstrate that these algorithms achieve superior performance compared to baseline approaches.

- For Plum-TS, although its performance is not the best, it achieves on-par performance as GrIPS in 4 times fewer API calls (as shown in Table 3). This advantage makes Plum-TS an extremely favorable choice for applications with limited budgets or API call quotas.

6 Conclusion

In this paper, a novel paradigm of prompt learning is proposed. By formulating prompt learning as a black-box discrete optimization problem, we are able to apply various metaheuristics algorithms to help discover effective prompts, while guaranteeing the whole process to be automatic, discrete, gradient-free, interpretable, and applicable to black-box models. Six typical Plum algorithms, Hill Climbing, Simulated Annealing, Genetic Algorithm (Mutation Only), Genetic Algorithm (with Crossover), Tabu Search, and Harmony Search all obtain non-trivial improvements over GrIPS on instruction following tasks. Furthermore, with the assistance of metaheuristics-based prompt learning, we are capable of discovering new patterns of effective prompts unexplored in the past.

Limitations

As an early step of bridging metaheuristics and prompt learning in a unified paradigm, we only implement six algorithms to prove the validity of this concept. Nevertheless, there are still a bunch of other potential metaheuristics that can be incorporated into this framework, producing more empirically useful methods for solving real-world problems. We hope Plum is just a starting point, more metaheuristics, such as Particle Swarm Optimization (Kennedy and Eberhart, 1995), Ant Colony Optimization (Dorigo and Gambardella, 1997) and Cuckoo search (Yang and Deb, 2009) can be combined with more neighborhood definitions, like retrieval-based completions and multimodal, to inspire more practical algorithms, which can eventually help us find a way to properly communicate with Large Language Models.

The potential risks of this work are the general risks of black prompt tuning methods, where more powerful prompt learning algorithms enable more powerful prompt attack approaches. Nevertheless, the silver lining is that most prompt attack methods normally in term generate adversarial samples that help improve the robustness of models, which can be considered beneficial in the long term.

References

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2023. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*.

Eyal Ben-David, Nadav Oved, and Roi Reichart. 2022. PADA: Example-based prompt learning for on-the-fly adaptation to unseen domains. *Transactions of the Association for Computational Linguistics*, 10:414–433.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Sondos Mahmoud Bsharat, Aidar Myrzakhan, and Zhiqiang Shen. 2023. Principled instructions are all you need for questioning llama-1/2, gpt-3.5/4. *arXiv preprint arXiv:2312.16171*.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, et al. 2023. Symbolic discovery of optimization algorithms. *arXiv preprint arXiv:2302.06675*.

Jiale Cheng, Xiao Liu, Kehan Zheng, Pei Ke, Hongning Wang, Yuxiao Dong, Jie Tang, and Minlie Huang. 2023. Black-box prompt optimization: Aligning large language models without model training. *arXiv preprint arXiv:2311.04155*.

WL Chiang, Z Li, Z Lin, Y Sheng, Z Wu, H Zhang, L Zheng, S Zhuang, Y Zhuang, JE Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, mar. 2023.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. *arXiv*.

Siddhartha Datta, Alexander Ku, Deepak Ramachandran, and Peter Anderson. 2023. Prompt expansion for adaptive text-to-image generation. *arXiv preprint arXiv:2312.16720*.

Joe Davison, Joshua Feldman, and Alexander Rush. 2019. Commonsense knowledge mining from pretrained models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1173–1178, Hong Kong, China. Association for Computational Linguistics.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric P Xing, and Zhiting Hu. 2022. Rlprompt: Optimizing discrete text prompts with reinforcement learning. *arXiv preprint arXiv:2205.12548*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Shizhe Diao, Zhichao Huang, Ruijia Xu, Xuechun Li, Yong Lin, Xiao Zhou, and Tong Zhang. 2022. Black-box prompt learning for pre-trained language models. *arXiv preprint arXiv:2201.08531*.

Shizhe Diao, Pengcheng Wang, Yong Lin, and Tong Zhang. 2023. Active prompting with chain-of-thought for large language models.

Hanze Dong, Wei Xiong, Deepanshu Goyal, Rui Pan, Shizhe Diao, Jipeng Zhang, Kashun Shum, and Tong Zhang. 2023. Raft: Reward ranked finetuning for generative foundation model alignment. *arXiv preprint arXiv:2304.06767*.

Marco Dorigo, Mauro Birattari, and Thomas Stutzle. 2006. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39.

Marco Dorigo and Luca Maria Gambardella. 1997. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1):53–66.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2021. Making pre-trained language models better few-shot learners. 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)*, pages 3816–3830, Online. Association for Computational Linguistics.

Zong Woo Geem, Joong Hoon Kim, and Gobichettipalayam Vasudevan Loganathan. 2001. A new heuristic optimization algorithm: harmony search. *simulation*, 76(2):60–68.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did Aristotle Use a Laptop? A Question Answering Benchmark with Implicit Reasoning Strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

Fred Glover. 1986. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549.

Fred Glover and Saıd Hanafi. 2002. Tabu search and finite convergence. *Discrete Applied Mathematics*, 119(1-2):3–36.

Peter Grabusts, Jurijs Musatovs, and Vladimir Golenkov. 2019. The application of simulated annealing method for optimal route detection between objects. *Procedia Computer Science*, 149:95–101.

Vincent Granville, Mirko Křivánek, and J-P Rasson. 1994. Simulated annealing: A proof of convergence. *IEEE transactions on pattern analysis and machine intelligence*, 16(6):652–656.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2023. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*.

Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. 2021. WARP: Word-level Adversarial ReProgramming. 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)*, pages 4921–4933, Online. Association for Computational Linguistics.

Xu Han, Weilin Zhao, Ning Ding, Zhiyuan Liu, and Maosong Sun. 2021. PTR: Prompt Tuning with Rules for Text Classification. *ArXiv preprint*, abs/2105.11259.

Adi Haviv, Jonathan Berant, and Amir Globerson. 2021. BERTese: Learning to speak to BERT. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 3618–3623, Online. Association for Computational Linguistics.

Xing-Shi He, Fan Wang, Yan Wang, and Xin-She Yang. 2018. Global convergence analysis of cuckoo search using markov theory. In *Nature-inspired algorithms and applied optimization*, pages 53–67. Springer.

John H Holland. 1992. Genetic algorithms. *Scientific american*, 267(1):66–73.

Bairu Hou, Joe O'Connor, Jacob Andreas, Shiyu Chang, and Yang Zhang. 2022. Promptboosting: Black-box text classification with ten forward passes. *arXiv preprint arXiv:2212.09257*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Abid Hussain, Yousaf Shad Muhammad, M Nauman Sajid, Ijaz Hussain, Alaa Mohamd Shoukry, Showkat Gani, et al. 2017. Genetic algorithm for traveling salesman problem with modified cycle crossover operator. *Computational intelligence and neuroscience*, 2017.

Kashif Hussain, Mohd Najib Mohd Salleh, Shi Cheng, and Yuhui Shi. 2019. Metaheuristic research: a comprehensive survey. *Artificial intelligence review*, 52(4):2191–2233.

Mojan Javaheripi and Sébastien Bubeck. 2024. Phi-2: The surprising power of small language models.

Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438.

James Kennedy and Russell Eberhart. 1995. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE.

Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. 1983. Optimization by simulated annealing. *science*, 220(4598):671–680.

Yuval Kirstain, Adam Polyak, Uriel Singer, Shahbuland Matiana, Joe Penna, and Omer Levy. 2023. Pick-a-pic: An open dataset of user preferences for text-to-image generation. *arXiv preprint arXiv:2305.01569*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157, San Diego, California. Association for Computational Linguistics.

Sawan Kumar and Partha Talukdar. 2021. Reordering examples helps during priming-based few-shot learning. *arXiv preprint arXiv:2106.01751*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. 2022. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021a. Gpt understands, too. *arXiv preprint arXiv:2103.10385*.

Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Kay Chen Tan. 2021b. A survey on evolutionary neural architecture search. *IEEE transactions on neural networks and learning systems*.

Marco Locatelli. 2000. Convergence of a simulated annealing algorithm for continuous global optimization. *Journal of Global Optimization*, 18(3):219–233.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2021. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv preprint arXiv:2104.08786*.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing English math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 975–984, Online. Association for Computational Linguistics.

Swaroop Mishra, Daniel Khashabi, Chitta Baral, Yejin Choi, and Hannaneh Hajishirzi. 2021. Reframing instructional prompts to gptk's language. *arXiv preprint arXiv:2109.07830*.

Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. 2022. Cross-task generalization via natural language crowdsourcing instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3470–3487.

OpenAI. 2023. Gpt-4 technical report.

Aziz Ouaarab, Belaïd Ahiod, and Xin-She Yang. 2014. Discrete cuckoo search algorithm for the travelling salesman problem. *Neural Computing and Applications*, 24(7):1659–1669.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Shilong Pan, Zhiliang Tian, Liang Ding, Zhen Huang, Zhihua Wen, and Dongsheng Li. 2024. Pomp: Probability-driven meta-graph prompter for llms in low-resource unsupervised neural machine translation.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP Models really able to Solve Simple Math Word Problems? *arXiv*.

Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. 2022. Grips: Gradient-free, edit-based instruction search for prompting large language models. *arXiv preprint arXiv:2203.07281*.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with" gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*.

Guanghui Qin and Jason Eisner. 2021. Learning how to ask: Querying LMs with mixtures of soft prompts. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5203–5212, Online. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Lothar M Schmitt. 2004. Theory of genetic algorithms ii: models for genetic operators over the string-tensor representation of populations and convergence to global optima for arbitrary fitness function under scaling. *Theoretical Computer Science*, 310(1-3):181–231.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235, Online. Association for Computational Linguistics.

KaShun Shum, Shizhe Diao, and Tong Zhang. 2023. Automatic prompt augmentation and selection with chain-of-thought from labeled data.

Hongjin Su, Jungo Kasai, Chen Henry Wu, Weijia Shi, Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, et al. 2022a. Selective annotation makes language models better few-shot learners. *ArXiv preprint*, abs/2209.01975.

Hsuan Su, Pohan Chi, Shih-Cheng Huang, Chung Ho Lam, Saurav Sahay, Shang-Tse Chen, and Hung-yi Lee. 2022b. Few-shot prompting toward controllable response generation. *arXiv preprint arXiv:2206.03931*.

Tianxiang Sun, Zhengfu He, Hong Qian, Yunhua Zhou, Xuan-Jing Huang, and Xipeng Qiu. 2022a. Bbtv2: Towards a gradient-free future with large language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3916–3930.

Tianxiang Sun, Yunfan Shao, Hong Qian, Xuanjing Huang, and Xipeng Qiu. 2022b. Black-box tuning for language-model-as-a-service. In *International Conference on Machine Learning*, pages 20841–20855. PMLR.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158, Minneapolis, Minnesota. Association for Computational Linguistics.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019. Universal adversarial triggers for attacking and analyzing NLP. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2153–2162, Hong Kong, China. Association for Computational Linguistics.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022a. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022b. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *arXiv*.

Hanwei Xu, Yujun Chen, Yulun Du, Nan Shao, Yanggang Wang, Haiyu Li, and Zhilin Yang. 2022. Gps: Genetic prompt search for efficient few-shot learning. *arXiv preprint arXiv:2210.17041*.

Xin-She Yang and Suash Deb. 2009. Cuckoo search via lévy flights. In *2009 World congress on nature & biologically inspired computing (NaBIC)*, pages 210–214. Ieee.

Jiahao Yu, Xingwei Lin, and Xinyu Xing. 2023. Gptfuzzer: Red teaming large language models with auto-generated jailbreak prompts. *arXiv preprint arXiv:2309.10253*.

Weizhe Yuan, Graham Neubig, and Pengfei Liu. 2021. BARTScore: Evaluating generated text as text generation. In *Advances in Neural Information Processing Systems*.

Biao Zhang, Ivan Titov, and Rico Sennrich. 2020a. Fast interleaved bidirectional sequence generation. In *Proceedings of the Fifth Conference on Machine Translation*, pages 503–515.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. 2020b. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International Conference on Machine Learning*, pages 11328–11339. PMLR.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. Tinyllama: An open-source small language model.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2022. Automatic chain of thought prompting in large language models.

Zexuan Zhong, Dan Friedman, and Danqi Chen. 2021. Factual probing is [MASK]: Learning vs. learning to recall. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5017–5033, Online. Association for Computational Linguistics.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*.

Kaijie Zhu, Qinlin Zhao, Hao Chen, Jindong Wang, and Xing Xie. 2023. Promptbench: A unified library for evaluation of large language models. *arXiv preprint arXiv:2312.07910*.

A Discovered Prompt Patterns

Text-to-Image Generation Several detailed prompts in text-to-image generation tasks are presented in Table 5. It can be observed that those improved prompts have a tendency to focus more on key descriptive elements in the image, such as “shadows”, “shimmer in petals” or “sunset”. The augmented prompts also use “There is” or “They have” to emphasize the main theme of the image and remove other distractive words.

Reasoning Here we present several typical cases of discovered prompt patterns for reasoning tasks, as shown in Table 6. It turns out that 1) completing the logical chain, 2) explaining the meaning of a term, or 3) providing additional clarification of a statement all help LLMs improve performance in reasoning tasks.

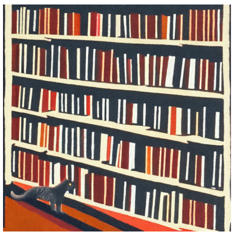
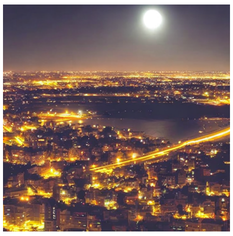
Topic	Original Image	Resultant Image	Original Prompt	Augmented Prompt
Cat in a library			A cat prowling in a library at night, books and shadows, silent observer.	A Is that true? Not at all cat prowling in shadows And books shadows,
Moonlight bathes blossom			Moonlight bathes blossom grove, petals shimmer in silvery luminescence	Moonlight bathes blossom silvery There is shimmer in petals
Sunflower			Sunflower field at sunset, golden hour, endless summer.	Sunflower endless field sunset, at sunset,
Moonlight bathes blossom			Moonlight illuminating at night, distant city lights.	Moonlight illuminating at night distant city They have lights
Glowing Mushroom Forest			Illuminated fungi cast ethereal light in dense woodland	Illuminated fungi cast It's something The and It is ethereal in dense

Table 5: `Plum-HS` augmented prompts for `stable-diffusion-2-1`

.

B Additional Experimental Results

B.1 Plum for White-Box Prompt Learning

More experimental results on Phi-2 (Javaheripi and Bubeck, 2024) and TinyLlama (Zhang et al., 2024) are shown in Table 7, where we still observe on-par or better performance when compare `Plum` with baseline methods. Additionally, Figure 2 illustrates the performance (accuracy) of `Plum-HS` on the Natural-Instructions dataset subtasks as the number of iterations increases, highlighting the effectiveness and efficiency of our proposed paradigm.

B.2 Plum for Chain-of-Thought

The `Plum` paradigm offers a natural approach to obtaining high-quality few-shot-CoT prompts for improving LLM's reasoning performances. To ensure that this class of prompts still remains logical and interpretable after optimization, we have made further improvements to the search process by introducing an additional LLM. Specifically, we encourage the language models to provide additional

Tasks	Searched Prompts
GSM8K	**Initial:** ··· If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot? ··· **GPT3:** ··· If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot (total number of cars in the parking lot + 2) ? ···
ASDiv	**Initial:** ··· Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total? ··· **GPT3.5:** ··· Leah had 32 chocolates and her sister had 42 (total number of chocolates owned by Leah and her sister) . If they ate 35 (total number of chocolates eaten by Leah and her sister) , how many pieces do they have left in total (total number of chocolates remaining with Leah and her sister) ? ···
AQuA	**Initial:** ··· How many keystrokes are needed to type the numbers from 1 to 500? Answer Choices: (a) 1156 (b) 1392 (c) 1480 (d) 1562 (e) 1788.··· **GPT3.5:** ··· How many keystrokes (the act of pressing a key on a keyboard) are needed to type the numbers from 1 to 500 (inclusive) ? Answer Choices: (a) 1156 (b) 1392 (c) 1480 (d) 1562 (e) 1788.···
CSQA	**Initial:** ··· What do people use to absorb extra ink from a fountain pen? Answer Choices: (a) shirt pocket (b) calligrapher's hand (c) inkwell (d) desk drawer (e) blotter ··· **GPT3.5:** ··· What do people use to absorb extra ink (liquid used for writing or printing) from a fountain pen (a pen that uses a reservoir of liquid ink to write) ? Answer Choices: (a) shirt pocket (b) calligrapher's hand (c) inkwell (d) desk drawer (e) blotter (a piece of absorbent material used to soak up excess ink or to dry freshly written ink) ···
StrategyQA	**Initial:** ··· Yes or no: Hydrogen's atomic number squared exceeds number of Spice Girls? ··· **GPT3.5:** ··· Yes or no: Does hydrogen have an atomic number of 1 (the number of protons in the nucleus of an atom) ?···

Table 6: Examples of initially given prompts and the corresponding searched prompts searched by `Plum`

Methods	Phi-2	TinyLlama
BDPL (Diao et al., 2022)	53.88±1.02	51.08±0.58
GrIPS (Prasad et al., 2022)	54.41±0.32	51.17±0.78
APO (Pryzant et al., 2023)	55.04±0.12	47.58±0.05
`Plum-HC`	54.46±1.55	51.71±0.77
`Plum-SA`	54.54±1.50	52.25±1.24
`Plum-GA-M`	53.67±0.79	51.13±1.24
`Plum-GA-C`	54.38±1.08	52.08±0.33
`Plum-TS`	54.25±1.89	**52.38±1.64**
`Plum-HS`	**55.25±0.91**	51.54±0.35

Table 7: Additional Experiments on white-box prompt learning tasks. BBT (Sun et al., 2022b)'s codebase is not fully compatible with Phi-2 and TinyLlama, hence the corresponding result is not available.

information and details in the `add` operation and introduce a teaching mechanism to guide the language models in revising prompts. On top of that, we empower the language models to autonomously split the prompts into phrases based on their own comprehension of the underlying text structure.

As depicted in Figure 3), these enhancements utilize the power of modern LLMs, aiming to accelerate the searching process of interpretable and target-model-friendly prompts via improved neighborhood definitions.

Dataset We test our methods on math word solving and commonsense reasoning tasks, including GSM8k (Cobbe et al., 2021), ASDiv (Miao et al., 2020), SVAMP (Patel et al., 2021), AQuA (Koncel-Kedziorski et al., 2016), CSQA (Talmor et al., 2019), and StrategyQA (Geva et al., 2021). Those tasks evaluate models' reasoning ability, where prompts are expected to elicit their logical thinking skills.

Experiment Setup `Plum` is applied to improving the performance of two backbones: GPT3 (text-curie-001) and GPT3.5 (gpt-turbo-3.5-0301) on aforementioned five reasoning tasks. Specifically, we employ `Plum-GA-M`* and `Plum-HS`* for this setting, majorly based on two reasons: 1) The effectiveness of both algorithms has been demonstrated in the previous general prompt learning setting, making them reasonable candidates for further extensions; 2) Compared with `Plum-GA-C`, both algorithms have lower computational complexity and similar exploration ability. Moreover, considering the API budgets and the extensive size of the test set for both math word solving and commonsense reasoning tasks, we chose to exclusively showcase the optimization performance of `Plum-GA-M`* and

	Prompting	GSM8K	ASDiv	SVAMP	AQuA	CSQA	StrategyQA
GPT3 (*text-curie-001*)	CoT	0.00	7.17	6.67	20.83	24.00	52.33
	`Plum-GA-M*`	4.66±0.47	9.67±1.25	11.67±0.94	22.00±0.82	29.00±0.82	61.00±1.63
	`Plum-HS*`	4.33±0.47	12.33±1.25	11.00±0.82	21.66±0.47	30.00±0.82	54.00±0.00
GPT3.5 (*gpt-3.5-turbo-0301*)	CoT	85.00	92.00	85.17	59.83	73.17	72.50
	`Plum-GA-M*`	87.33±0.94	94.67±0.94	88.00±0.82	67.67±1.25	77.33±0.47	78.00±1.41
	`Plum-HS*`	85.33±0.47	92.33±0.47	87.00±0.82	67.67±1.89	77.00±0.82	73.00±0.82

Table 8: Improvements of optimization performance for the few-shot-CoT prompts searched by `Plum-GA-M*` and `Plum-HS*` as the searching strategies on math word solving and commonsense reasoning tasks.

`Plum-HS*` on the training sets for reference.

Results In this section, we present the enhancement achieved by the few-shot-CoT prompts searched by `Plum` across five demanding downstream tasks, as shown in Table 8. These results clearly demonstrate the superiority of `Plum-GA-M*` and `Plum-HS*` over standard few-shot-CoT prompts used for GPT3 and GPT3.5. In particular, our methods achieve universal improvements across those five tasks on almost all the backbone models.

In addition, despite the already impressive performance of GPT3.5 with standard few-shot-CoT prompts, we continue to observe even greater average improvements compared to GPT3. This is due to two key factors. Firstly, the edit operations we have developed in this section allow GPT3.5 to generate better prompts for itself. Secondly, GPT3.5 exhibits stronger capabilities in comprehending and leveraging the enhanced prompts, further improving its overall performance.

Experimental Details The initial few-shot-CoT prompts provided followed the **Instruction + Examples** format, consisting of a task-agnostic instruction and several positive examples (details are shown in Figure 4, which is consistent with the setting described in (Prasad et al., 2022). The **Examples** part of the initial prompts is the same as the few-shot exemplars by Wei et al. (2022b). The **Instruction** provides a general description of the downstream task, although its specific content may vary depending on the backbone models used. The new prompt candidates are generated by applying edits to both the instruction and examples with equal probability during the searching process, i.e $Prob\{\text{edit the instruction}\} = Prob\{\text{edit the examples}\} = 1/2$.

For GPT3, the **Instruction** part of the prompt follows the zero-shot-CoT approach proposed by Kojima et al. (2022). Meanwhile, for GPT3.5, the **Instruction** part of the initial prompts is manually crafted by referring to the API reference documentation available on the GPT3.5 in the OpenAI Platform[1].

Moreover, Figure 5 illustrates the template for the prompt used of the improved add operation and revisions, leveraging the GPT3.5 as the underlying model.

Experimental Cost While producing few-shot-CoT prompts, we do not consider the API call limit; however, we still take into account the estimated cost of the experiments. The costs for GPT3 (*text-curie-001*) and GPT3.5 (*gpt-3.5-turbo-0301*) are nearly identical as they share the same pricing structure ($0.002 per 1k tokens). On average, each task typically involves a cost of approximately $25 and concludes after approximately 12 hours of execution when searching for few-shot-CoT prompts with a single thread.

C Algorithm Details

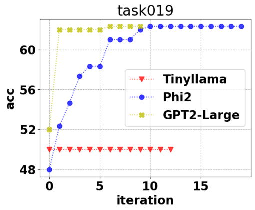
To help users better understand and utilize the package, here we present the main idea and pseudocodes of all included algorithms so far.

C.1 Hill Climbing

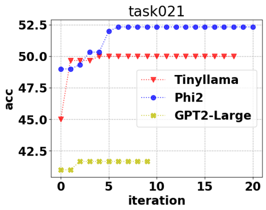
Hill climbing is a greedy algorithm that aims to find the local optimum of a given objective, e.g. the score function. Each time the algorithm explores a fraction of possible prompt changes and selects one that improves the score.

In our version, we select the one with the most improvements to further speed up the optimization process, as shown in Algorithm 2. Notice that in the paradigm of EvoPrompt, GrIPS can also be viewed as a special version of Hill Climbing, given it greedily improves the prompt via edit operations.

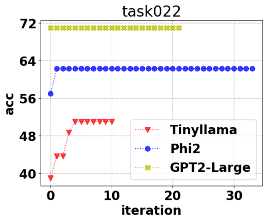
[1] https://platform.openai.com/docs/guides/gpt/chat-completions-api



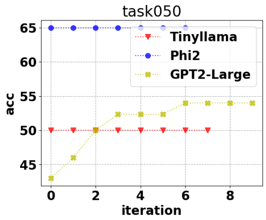
(a) Task 019



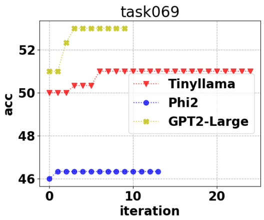
(b) Task 021



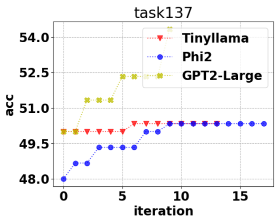
(c) Task 022



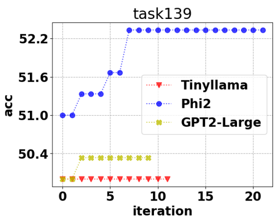
(d) Task 050



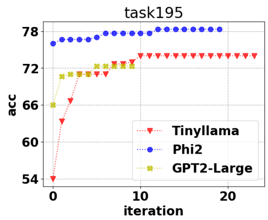
(e) Task 069



(f) Task 137



(g) Task 139



(h) Task 195

Figure 2: Performance of `Plum-HS` across different iterations on subtasks of Natural-Instructions dataset.

C.2 Simulated Annealing

Different from hill climbing, simulated annealing (Kirkpatrick et al., 1983) allows a small probability of deviation when the prompt change does not yield instant improvement, thus capable of escaping from local optimums and finding better solutions. The probability of deviation is controlled by a hyperparameter called "temperature" $T(i)$ at iteration i, where the lower temperature is, smaller the deviation probability is, less random the algorithm behavior becomes. When the temperature $T(i) = 0$, simulated annealing becomes deterministic and degenerates to hill climbing.

As shown in Algorithm 3, it only differs from hill climbing in one line, where it utilizes the provided temperature $T(i)$ to decide the deviation probability. We use exponential decay $T(i) = 10\exp(-i/5)$ as our default temperature schedule in the package.

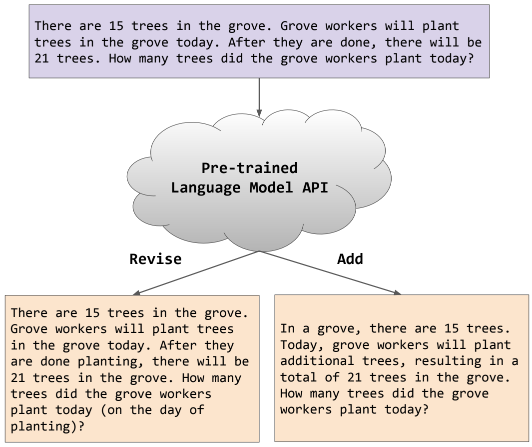


Figure 3: Illustration of the improved add operation and revision by language models

Algorithm 2 GrIPS / Hill Climbing

$base \leftarrow init$ ▷ Initialize base candidate
$s_{base} \leftarrow \texttt{score}(base)$ ▷ Score using examples in $\mathcal{S}$
$\Omega \leftarrow \{\texttt{del}, \texttt{swap}, \texttt{par}, \texttt{add}\}$ ▷ Set of edit operations
$\rho \leftarrow P$ ▷ Patience for early-stop
for $i = 1, \cdots, n$ **do** ▷ n: number of iterations
  for $j = 1, \cdots, m$ **do** ▷ m: number of candidates
    Sample $e_1, \cdots, e_l \in \Omega$ ▷ l edits per candidate
    $\mathcal{C}[j] \leftarrow \texttt{edit}(base, e_1 \circ \cdots \circ e_l)$
    $s[j] \leftarrow \texttt{score}(\mathcal{C}[j])$ ▷ Score above candidate
  $k \leftarrow \operatorname{argmax}_j s[j]$
  $best \leftarrow \mathcal{C}[k]$ ▷ Chosen Candidate
  $s_{best} \leftarrow s[k]$ ▷ Score of chosen candidate
  if $s_{best} > s_{base}$ **then** ▷ Candidate better than base
    $base \leftarrow best$ ▷ Use this candidate in the next step
    $s_{base} \leftarrow s_{best}$ ▷ Update base score
    $\rho \leftarrow P$ ▷ Refresh patience
  else
    if $\rho > 0$ **then** ▷ Patience not exhausted
      decrement ρ
      continue ▷ Continue search with same base
    else
      return base ▷ Early-stop criteria met
return base ▷ Search terminates after the last iteration

C.3 Genetic Algorithm (Mutation Only)

Genetic algorithm (Holland, 1992) was inspired by biological evolution during the nature selection process, where each prompt is treated as a "gene" and the score function serves as the criterion of nature selection.

In a simplified version of genetic algorithm, as illustrated in Algorithm 4, only mutation operations are adopted to change the prompt. A population of prompts $\mathbf{W}$ are maintained during the optimization process, indicating all the possible exploring directions of the algorithm. At each iteration, we select a high-quality prompt from all those exploring direc-

Instruction: Let's think step by step.

Examples:

Q: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?

A: There are 15 trees originally. Then there were 21 trees after some more were planted. So there must have been 21 - 15 = 6. The answer is 6.

Input:

Q: Omi is twice as old as Kimiko. Arlette is 3/4 times as old as Kimiko. If Kimiko is 28 years old, calculate the average age of the three?

A:

Instruction:

{"role":"system","content": You are an assistant proficient in mathematical calculations. Solve the math word problems below, showing the reasoning process and results. Here are some examples.}

Examples:

{"role":"user","content": There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?}

{"role":"assistant","content": There are 15 trees originally. Then there were 21 trees after some more were planted. So there must have been 21 - 15 = 6. The answer is 6.}

Input:

{"role":"user","content": Omi is twice as old as Kimiko. Arlette is 3/4 times as old as Kimiko. If Kimiko is 28 years old, calculate the average age of the three?}

Figure 4: Illustration of the initial prompt for GSM8K with GPT3 **(top)** and GPT3.5 **(bottom)** as backbone model

Instruction:

{"role":"system","content": You are a helpful English writing assistant. Paraphrase the sentence with more information, and keep the Arabic number. Do not attempt to answer the question. Here are three examples.}

Examples:

{"role":"user","content": At the duck park, there are 25 mallard ducks and ten less than twice as many geese as ducks. Then, a small flock of 4 ducks arrived at the park and joined the birds already there. If five less than 15 geese leave the park, how many more geese than ducks remain at the park?}

{"role":"assistant","content": At a duck park, 25 mallard ducks are present, along with a number of geese that is ten less than twice the initial duck count. Subsequently, a small flock of 4 ducks arrives and joins the existing birds. If a group of geese departs, leaving 5 fewer than 15 geese. How many more geese than ducks remain in the park?}

Input:

{"role":"user","content": Omi is twice as old as Kimiko. Arlette is 3/4 times as old as Kimiko. If Kimiko is 28 years old, calculate the average age of the three?}

Instruction:

{"role":"system","content": You are a helpful English writing assistant. Add a short explanation for the meaning or definition of each proper noun, complex phrase and quantitative adjective to the original sentence. Here are four examples.}

Examples:

{"role":"user","content": At a flea market, Hillary sells handmade crafts for 12 dollars per craft. Today, Hillary sells 3 crafts and is given an extra 7 dollars from an appreciative customer. Later on, Hillary deposits half of today's profits into her bank account. How many dollars is Hillary left so far? Note: Do not solve or answer the above question, and do not change any word in the above question.}
{"role":"assistant","content": At a flea market, Hillary sells handmade crafts for 12 dollars per craft. Today, Hillary sells 3 crafts and is given an extra 7 dollars from an appreciative customer. Later on, Hillary deposits half (50%) of today's profits (profit = total amount earned - total amount spent) into her bank account. How many dollars is Hillary left so far (until now)?}

Input:

{"role":"user","content": Omi is twice as old as Kimiko. Arlette is 3/4 times as old as Kimiko. If Kimiko is 28 years old, calculate the average age of the three? Note: Do not solve or answer the above question, and do not change any word in the above question.}

Figure 5: Illustration of the improved add operation and revision with GPT3.5 as backbone model

tions via tournament selection. Intuitively, the more tournament round k we adopt, the higher quality of the selected prompt, but the less explorative the algorithm behavior becomes. The selected prompt is then mutated to generate a new prompt.

As the algorithm proceeds, the average quality of prompts in the population is expected to improve. In fact, given sufficient number of iterations, the algorithm is capable of finding the global optimum.

C.4 Genetic Algorithm (with Crossover)

The general version of genetic algorithm (Holland, 1992) is realized in Algorithm 5. Besides the mutation operation, a different operation called "crossover" is introduced, where we allow new prompts to be generated from two "parent prompts". The new prompts will inherit partial segments from both parent prompts and be further changed with mutation operations.

Intuitively speaking, crossover operation represents "global" changes in the search space, while mutation operation represents "local" changes, hence it is to be expected that the genetic algorithm with crossover is less efficient than its simplified counterpart, but more explorative and capable of finding better solutions with sufficient iterations.

C.5 Tabu Search

Greedy search algorithms tend to be stuck in a local optimum or near a local optimum. Tabu Search (Glover, 1986) effectively compensates for this shortcoming by introducing the Tabu list: any candidate that was visited recently is forbidden to visit again. This allows the optimization process to leave the local "tabu" area and explore more.

In Algorithm 6 we adopt a simple Tabu rule of direct comparison: with a high probability, a generated prompt that exactly matches any prompts in the Tabu list will be treated as invalid and discarded, where the Tabu list only retains the N_{tabu} latest generated prompts.

C.6 Harmony Search

Harmony Search (Geem et al., 2001) mimics the improvision procedure of musicians, where a new prompt is generated via combining segments of all stored prompts in the harmony memory. A pitch finetuning process is then conducted, allowing further refinement of the prompt. The full details are available in Algorithm 7.

Algorithm 3 Simulated Annealing

$base \leftarrow init$ ▷ Initialize base candidate
$s_{base} \leftarrow \texttt{score}(base)$ ▷ Score using examples in $\mathcal{S}$
$\Omega \leftarrow \{\texttt{del}, \texttt{swap}, \texttt{par}, \texttt{add}\}$ ▷ Set of edit operations
$\rho \leftarrow P$ ▷ Patience for early-stop
for $i = 1, \cdots, n$ **do** ▷ n: number of iterations
  for $j = 1, \cdots, m$ **do** ▷ m: number of candidates
    Sample $e_1, \cdots, e_l \in \Omega$ ▷ l edits per candidate
    $\mathcal{C}[j] \leftarrow \texttt{edit}(base, e_1 \circ \cdots \circ e_l)$
    $s[j] \leftarrow \texttt{score}(\mathcal{C}[j])$ ▷ Score above candidate
  $k \leftarrow \text{argmax}_j\, s[j]$
  $best \leftarrow \mathcal{C}[k]$ ▷ Chosen Candidate
  $s_{best} \leftarrow s[k]$ ▷ Score of chosen candidate
  if $s_{best} > s_{base}$ **or** $\exp(\frac{s_{best} - s_{base}}{T(i)}) \geq \texttt{Random}(\mathcal{U}(0,1))$ **then** ▷ Candidate better than base
    $base \leftarrow best$ ▷ Use this candidate in the next step
    $s_{base} \leftarrow s_{best}$ ▷ Update base score
    $\rho \leftarrow P$ ▷ Refresh patience
  else
    if $\rho > 0$ **then** ▷ Patience not exhausted
      decrement ρ
      continue ▷ Continue search with same base
    else
      return base ▷ Early-stop criteria met
return base ▷ Search terminates after the last iteration

Algorithm 4 Genetic Algorithm (mutation only)

$base \leftarrow init$ ▷ Initialize base candidate
$s_{base} \leftarrow \texttt{score}(base)$ ▷ Score using examples in $\mathcal{S}$
$\Omega \leftarrow \{\texttt{del}, \texttt{swap}, \texttt{par}, \texttt{add}\}$ ▷ Set of edit operations
$\rho \leftarrow P$ ▷ Patience for early-stop
$\mathbf{W} \leftarrow \{\langle base, s_{base} \rangle\}$
$result \leftarrow base$ ▷ Initialize population
for $i = 1, 2, \cdots, n$ **do** ▷ n: number of iterations
  $\mathbf{S} \leftarrow \emptyset$ ▷ Examples that are going to battle in the tournament
  for $j = 1, 2, \cdots, k$ **do** ▷ k: tournament selection hyperparameter
    $\langle parent, s_{parent} \rangle \leftarrow \texttt{Random}(\mathbf{W})$
    $\mathbf{S} \leftarrow \mathbf{S} + \{\langle parent, s_{parent} \rangle\}$
  $base \leftarrow \text{argmax}_{\{parent | \langle parent, s_{parent} \rangle \in \mathbf{S}\}} s_{parent}$
  for $j = 1, \cdots, m$ **do** ▷ m: number of candidates
    Sample $e_1, \cdots, e_l \in \Omega$ ▷ l edits per candidate
    $\mathcal{C}[j] \leftarrow \texttt{edit}(base, e_1 \circ \cdots \circ e_l)$
    $s[j] \leftarrow \texttt{score}(\mathcal{C}[j])$ ▷ Score above candidate
  $k \leftarrow \text{argmax}_j\, s[j]$
  $best \leftarrow \mathcal{C}[k]$ ▷ Chosen Candidate
  $s_{best} \leftarrow s[k]$ ▷ Score of chosen candidate
  $\mathbf{W} \leftarrow \mathbf{W} + \{\langle best, s_{best} \rangle\}$
  if $s_{best} > s_{result}$ **then** ▷ Candidate better than best result so far
    $result \leftarrow best$
    $s_{result} \leftarrow s_{best}$
    $\rho \leftarrow P$ ▷ Refresh patience
  else
    if $\rho > 0$ **then** ▷ Patience not exhausted
      decrement ρ
      continue ▷ Continue search with same base
    else
      return $result$ ▷ Early-stop criteria met
return $result$ ▷ Search terminates after the last iteration

D Hyperparameter Setting

For all the experiments, the number of edits that performs on the candidate prompts ($num\_compose$) and the number of the newly generated candidates ($num\_candidate$) in each iteration follows $num\_compose \in \{1, 2\}$ and $num\_candidate \in \{5, 10\}$. The population sizes, and tournament selections k of Plum-GA-M, Plum-GA-C and Plum-GA-M* are all set to 10 and 3 respectively. Meanwhile, the mutation rate p_{mutation} is set to 0.5 for Plum-GA-C. The N_{tabu} is configured to 5 for Plum-TS. Additionally, the configuration for the harmony search memory (N_H), the number of segments (k_s), harmony memory considering rate ($HMCR$), and pitching adjust rate (PAR) of Plum-HS and Plum-HS* is as follows: 10, 5, 0.4, and 0.5, respectively. Furthermore, the temperatures of GPT-3 are configured to 0 in our experiments.

In the context of general prompt learning, disregarding the API call limit allows us to set the maximum iteration and patience parameters to 50 and 7 respectively, which ensures that Plum-GA-M and Plum-GA-C have a sufficient number of iterations to generate relatively optimal solutions.

When it comes to few-shot-CoT prompt learning, setting a very large value for the maximum iterations is not recommended due to the potential cost implications, which is primarily because the number of tokens generated in each iteration tends to be substantial. And for Plum-GA-M* and Plum-HS*, effective results can be obtained with maximum iterations ranging from 5 to 15, coupled with a patience level ranging from 2 to 5. For the experimental results shown in Table 8, we configured the maximum iterations to be 10 and deactivated the patience parameter.

For text-to-image generation tasks, to accommodate a wide range of application scenarios, the initial prompts for image generation are kept short (merely 5-15 words). Given that stable diffusion requires approximately 15 seconds (on a NVIDIA A40 GPU) to produce each image, and the scoring process necessitates the generation of two images for direct comparison, it has been observed that after 6-7 iterations, most prompts reach a point of convergence where the best score is achieved. Consequently, it is advisable to limit the number of iterations to 5-15 and set the patience parameter between 2-5. In the experimental results presented in Table 5, we set the maximum iterations to 10 and deactivated the patience parameter. Further-

Algorithm 5 Genetic Algorithm (with crossover)

Hyperparameters: Number of iterations n, population size N_p, number of offsprings each generation $N_{\text{offspring}}$, mutation probability $p_{mutation}$.

$base \leftarrow init$ ▷ Initialize base candidate
$s_{base} \leftarrow \texttt{score}(base)$ ▷ Score using examples in $\mathcal{S}$
$\Omega \leftarrow \{\texttt{del}, \texttt{swap}, \texttt{par}, \texttt{add}\}$ ▷ Set of edit operations
$\rho \leftarrow P$ ▷ Patience for early-stop
$\mathbf{W}_0 \leftarrow \{\langle base, s_{base}\rangle \times N_p\}$ ▷ N_p copies of the initial prompt
$result \leftarrow base$
for $i = 1, 2, \cdots, n$ **do** ▷ n: number of iterations
  $\mathbf{W}_i \leftarrow \mathbf{W}_{i-1}$
  for $j = 1, 2, \cdots, N_{\text{offspring}}$ **do** ▷ ===== Crossover
    $\langle parent_1, s_{parent_1}\rangle \leftarrow$ BINARY_TOURNAMENT_SELECTION($\mathbf{W}_{i-1}$)
    $\langle parent_2, s_{parent_2}\rangle \leftarrow$ BINARY_TOURNAMENT_SELECTION($\mathbf{W}_{i-1}$)
    offspring $\leftarrow$ CROSSOVER($parent_1, parent_2$)
    if offspring $\notin \mathbf{W}_i$ **then**
      $s_{\text{offspring}} \leftarrow \texttt{score}(\text{offspring})$
      $\mathbf{W}_i \leftarrow \mathbf{W}_i + \{\langle \text{offspring}, s_{\text{offspring}}\rangle\}$
  $\mathbf{W}_i \leftarrow \texttt{top-}N_p(\mathbf{W}_i)$ ▷ Removes prompts with bad scores, reserves only best N_p prompts
  $\rho, result, flag_{stop} \leftarrow$ UPDATE_RESULT($\rho, result, \mathbf{W}_i$)
  if $flag_{stop}$ **then return** $result$

  $\mathbf{W}'_i \leftarrow \mathbf{W}_i$
  for $\forall \langle base, s_{base}\rangle \in \mathbf{W}_i$ **do** ▷ ===== Mutation
    if $p_{mutation} \geq \texttt{Random}([0,1])$ **then**
      Sample $e_1, \cdots, e_l \in \Omega$ ▷ l edits per candidate
      $mutated \leftarrow \texttt{edit}(base, e_1 \circ \cdots \circ e_l)$
      $s_{mutated} \leftarrow \texttt{score}(mutated)$ ▷ Score above candidate
      $\mathbf{W}'_i \leftarrow \mathbf{W}'_i + \{\langle mutated, s_{mutated}\rangle\}$
  $\mathbf{W}_i \leftarrow \texttt{top-}N_p(\mathbf{W}'_i)$ ▷ Removes prompts with bad scores, reserves only best N_p prompts
  $\rho, result, flag_{stop} \leftarrow$ UPDATE_RESULT($\rho, result, \mathbf{W}_i$)
  if $flag_{stop}$ **then return** $result$
return $result$ ▷ Search terminates after last iteration

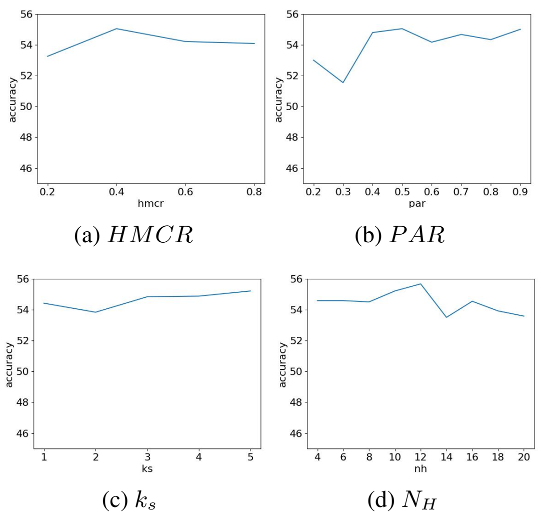
function BINARY_TOURNAMENT_SELECTION($\mathbf{W}$)
  $\mathbf{S} \leftarrow \emptyset$
  for $i = 1, 2$ **do**
    $\langle parent, s_{parent}\rangle \leftarrow \texttt{Random}(\mathbf{W})$
    $\mathbf{S} \leftarrow \mathbf{S} + \{\langle parent, s_{parent}\rangle\}$
  return $\text{argmax}_{\{parent|\langle parent, s_{parent}\rangle \in \mathbf{S}\}} s_{parent}$

function CROSSOVER($parent_1, parent_2$)
  $[\mathbf{t}_{1,1}, \mathbf{t}_{1,2}, \ldots, \mathbf{t}_{1,L_1}] \leftarrow parent_1$
  $[\mathbf{t}_{2,1}, \mathbf{t}_{2,2}, \ldots, \mathbf{t}_{2,L_2}] \leftarrow parent_2$
  $split \leftarrow \texttt{Random}(\{0, 1, 2, \ldots, \max(L_1, L_2)\})$
  offspring $\leftarrow [\mathbf{t}_{1,1}, \ldots, \mathbf{t}_{1,split}, \mathbf{t}_{2,split+1}, \ldots, \mathbf{t}_{2,L_2}]$
  return offspring

function UPDATE_RESULT($\rho, result, \mathbf{W}$)
  $flag_{stop} \leftarrow$ **false**
  $best \leftarrow \text{argmax}_{\{prompt|\langle prompt, s_{prompt}\rangle \in \mathbf{W}\}} s_{prompt}$
  if $s_{best} > s_{result}$ **then** ▷ Candidate better than best result so far
    $result \leftarrow best$
    $s_{result} \leftarrow s_{best}$
    $\rho \leftarrow P$ ▷ Refresh patience
  else
    if $\rho > 0$ **then** ▷ Patience not exhausted
      decrement ρ
    else
      $flag_{stop} \leftarrow$ **true** ▷ Early-stop criteria met
  return $\rho, result, flag_{stop}$

Algorithm 6 Tabu Search

$base \leftarrow init$ ▷ Initialize base candidate
$s_{base} \leftarrow \texttt{score}(base)$ ▷ Score using examples in $\mathcal{S}$
$\Omega \leftarrow \{\texttt{del}, \texttt{swap}, \texttt{par}, \texttt{add}\}$ ▷ Set of edit operations
$\rho \leftarrow P$ ▷ Patience for early-stop
$result \leftarrow base$ ▷ Initialize population
$w_0 \leftarrow base$
$\mathcal{T} \leftarrow \{w_0\}$ ▷ Initialize $\mathcal{T}$
for $i = 1, 2, \cdots, K$ **do** ▷ n: number of iterations
  for $j = 1, 2, \cdots, n$ **do** ▷ k: number of the new candidate
    Sample $e_1, \cdots, e_l \in \Omega$ ▷ l edits per candidate
    $\mathcal{C}[j] \leftarrow \texttt{edit}(base, e_1 \circ \cdots \circ e_l)$
    $s[j] \leftarrow \texttt{score}(\mathcal{C}[j])$ ▷ Score above candidate
  $\mathcal{W}'_j \leftarrow$ TABU($\mathcal{T}, s[j], TEMP$) = 0
  $k \leftarrow \text{argmax}_{l \in \mathcal{W}'_j} s[j]$
  $best \leftarrow \mathcal{C}[k]$ ▷ Chosen Candidate
  $s_{best} \leftarrow s[k]$ ▷ Score of chosen candidate
  $\mathcal{T} \leftarrow \mathcal{T} + s[k]$ ▷ Update $\mathcal{T}$
  if $s_{best} > s_{result}$ **then** ▷ Candidate better than best result so far
    $result \leftarrow best$
    $s_{result} \leftarrow s_{best}$
    $\rho \leftarrow P$ ▷ Refresh patience
  else
    if $\rho > 0$ **then** ▷ Patience not exhausted
      decrement ρ
      continue ▷ Continue search with same base
    else
      return $result$ ▷ Early-stop criteria met
  if $|\mathcal{T}| > N_{tabu}$ **then**
    $\mathcal{T} \leftarrow \mathcal{T} - w_{k-N_{tabu}}$ ▷ Keep Short-term tabus
return $result$ ▷ Search terminates after last iteration
function TABU($\mathcal{T}$,w, $TEMP$)
  if $w \in \mathcal{T}$ **then**
    if $TEMP \geq \texttt{Random}(\mathcal{U}(0,1))$ **then**
      return 0
    else
      return 1
  else
    return 0



(a) $HMCR$ (b) PAR (c) k_s (d) N_H

Figure 6: The effects of the parameters of `Plum-HS` with GPT2-large as backbone.

more, considering the prompt's brevity, the initial k_s value was set to 2 in the experiments to guarantee that each segment targeted for modification includes at least one tag. Additionally, the Harmony Memory Considering Rate ($HMCR$), Pitch Adjustment Rate (PAR), and Harmony Search Memory (n_H) were set to 0.4, 0.5, and 10 respectively.

Sensitiveness of `Plum-HS` Hyperparameters To gain deeper insights into the impact of key hyperparameters in Harmony Search and guide their effective tuning, we conducted an ablation study on $HMCR$, k_s and N_H for `Plum-HS` with GPT2-large as the backbone model. As shown in Figure 6, both the Harmony Memory Consideration

Algorithm 7 Harmony Search

$base \leftarrow init$ ▷ Initialize base candidate
$s_{base} \leftarrow \texttt{score}(base)$ ▷ Score using examples in $\mathcal{S}$
$\Omega \leftarrow \{\texttt{del}, \texttt{swap}, \texttt{par}, \texttt{add}\}$
$\Omega_{small} \leftarrow \{\texttt{par}\}$ ▷ Set of edit operations
$\rho \leftarrow P$ ▷ Patience for early-stop
$\mathbf{W}_0 \leftarrow \{\langle base, s_{base}\rangle\}$
$result \leftarrow base$ ▷ Initialize population
for $i = 1, 2, \cdots, n$ **do** ▷ n: number of iterations
  $\mathbf{W}' \leftarrow \emptyset$ ▷ Candidates generated at this iteration
  for $c = 1, 2, \cdots, k$ **do**
    $\mathbf{w}' \leftarrow$ GENERATE_CANDIDATE$(\mathbf{W}_{i-1}, \Omega_{small}, \Omega)$
    $s_{\mathbf{w}'} \leftarrow \texttt{score}(\mathbf{w}')$ ▷ Score the new candidate
    $\mathbf{W}' \leftarrow \mathbf{W}' + \{\langle \mathbf{w}', s_{\mathbf{w}'}\rangle\}$
  $\rho, result, flag_{stop} \leftarrow$ UPDATE_RESULT$(\rho, result, \mathbf{W}')$
  if $flag_{stop}$ **then return** $result$
  $\mathbf{W}_i \leftarrow \texttt{top-}N_H(\mathbf{W}_{i-1} \cup \mathbf{W}')$ ▷ Removes prompts with bad scores, reserves only best N_H prompts
return $result$ ▷ Search terminates after last iteration
function GENERATE_CANDIDATE$(\mathbf{W}, \Omega_{small}, \Omega)$
  $\mathbf{w}' \leftarrow [\,]$ ▷ Empty string/list
  for $j = 1, 2, \cdots, k_s$ **do** ▷ k_s: number of segments (pitches)
    $\langle \mathbf{w}, *\rangle \leftarrow \texttt{Random}(\mathbf{W})$ ▷ Randomly sample an existing prompt for j-th segment
    $L \leftarrow |\mathbf{w}|$ ▷ Length of the reference prompt
    $[w_0, w_1, \ldots, w_{L-1}] \leftarrow \mathbf{w}$
    $start \leftarrow \lceil \frac{j-1}{k_s} \cdot L \rceil$
    $end \leftarrow \lceil \frac{j}{k_s} \cdot L \rceil - 1$
    $\mathbf{w}_{segment} \leftarrow [w_{start}, w_{start+1}, \ldots, w_{end}]$
    if $HMCR \geq \texttt{Random}(\mathcal{U}(0,1))$ **then**
      if $PAR \geq \texttt{Random}(\mathcal{U}(0,1))$ **then** ▷ A little different random segment (pitch)
        Sample $e_1, \cdots, e_l \in \Omega_{small}$ ▷ Only replace phrases with their synonyms
        $\mathbf{w}_{segment} \leftarrow \texttt{edit}(\mathbf{w}_{segment}, e_1 \circ \cdots \circ e_l)$
    else ▷ A largely different random segment (pitch)
      Sample $e_1, \cdots, e_l \in \Omega$ ▷ A big change in the original segment
      $\mathbf{w}_{segment} \leftarrow \texttt{edit}(\mathbf{w}_{segment}, e_1 \circ \cdots \circ e_l)$
    $\mathbf{w}' = \mathbf{w}' + \mathbf{w}_{segment}$ ▷ Here "+" means concatenation.
  return $\mathbf{w}'$
function UPDATE_RESULT$(\rho, result, \mathbf{W})$
  $flag_{stop} \leftarrow$ **false**
  $best \leftarrow \text{argmax}_{\{prompt | \langle prompt, s_{prompt}\rangle \in \mathbf{W}\}} s_{prompt}$
  if $s_{best} > s_{result}$ **then** ▷ Candidate better than best result so far
    $result \leftarrow best$
    $s_{result} \leftarrow s_{best}$
    $\rho \leftarrow P$ ▷ Refresh patience
  else
    if $\rho > 0$ **then** ▷ Patience not exhausted
      decrement ρ
    else
      $flag_{stop} \leftarrow$ **true** ▷ Early-stop criteria met
  return $\rho, result, flag_{stop}$

Rate $HMCR$ and the Harmony Memory size N_H achieve optimal performance with a moderate value. This trend aligns with the core principles of Harmony Search (Algorithm 7). Low $HMCR$ encourages the algorithm perform random modifications, which results in introducing too much difference. Conversely, high $HMCR$ over-relies on past prompts in the harmony memory, which lacks diversity. As for Harmony Memory size N_H, it is similar to the population size in genetic algorithms. Small N_H restricts exploration such that the algorithm behaves similarly to greedy search and lacks diversity. While, large N_H deteriorates the quality of memorized prompts, and slows down the search efficiency. Notably, the performance exhibits stable and high performance when PAR is set to 0.4 or above.

For k_s, we find that a larger value introduces superior performance. This is because larger k_s leads to finer granularity of segments, which allows a finer level of tuning on prompts. In this paper, the upper bound of k_s is considered as 5 since the shortest sample in our datasets only contains 5 segments of words.

E Licenses

For general prompt learning tasks, the dataset Natural-Instructions v2.6 (Mishra et al., 2022) is released under Apache-2.0 license. While for Chain-of-Thought prompt learning, the datasets ASDiv (Miao et al., 2020), SVAMP (Patel et al., 2021), CSQA (Talmor et al., 2019) are released under CC-BY-NC 4.0, MIT, and CC BY-SA 4.0 licenses respectively.