Automatic Engineering of Long Prompts

Cho-Jui Hsieh, Si Si, Felix X. Yu, Inderjit S. Dhillon
Google Inc.
{cjhsieh, sisidaisy, felixyu, isd}@google.com

Abstract

Large language models (LLMs) have demonstrated remarkable capabilities in solving complex open-domain tasks, guided by comprehensive instructions and demonstrations provided in the form of prompts. However, these prompts can be lengthy, often comprising hundreds of lines and thousands of tokens, and their design often requires considerable human effort. Recent research has explored automatic prompt engineering for short prompts, typically consisting of one or a few sentences. However, the automatic design of long prompts remains a challenging problem due to its immense search space. In this paper, we propose an algorithm named Automated Prompt Engineering Xpert (APEX), a novel algorithm that automatically improves long prompts. Leveraging a greedy algorithm with beam-search for efficiency, APEX utilizes search history to significantly enhance the effectiveness of LLM-based mutation in its search process. Our results show that APEX achieves an average of 9.2% accuracy gain on eight tasks in Big Bench Hard and a consistent improvements on GSM8K with various models, highlighting the significance of automating prompt designs to fully harness the capabilities of LLMs.

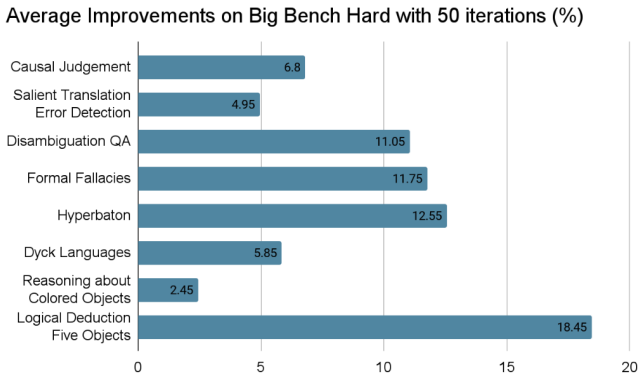
1 Introduction

Large language models (LLMs) have exhibited remarkable capabilities when trained on large datasets, demonstrating the ability to comprehend complex and lengthy instructions for diverse tasks without the need for fine-tuning (Wei et al., 2022a; Brown et al., 2020; Chowdhery et al., 2022; Ouyang et al., 2022). Several prompt design principles have emerged in recent years, suggesting that incorporating more complex instructions, demonstrations, and chain-of-thought reasoning into prompts can boost the performance on challenging tasks (Brown et al., 2020; Wei et al., 2022b), including those involving mathematical problem-solving (Cobbe et al., 2021) and reasoning (Suzgun et al., 2022; Srivastava et al., 2022). However, effective prompts for tackling complex tasks often contain thousands of tokens, posing challenges in designing and optimizing them. Figure 1 demonstrates a long prompt for a task in Big Bench Hard (Suzgun et al., 2022), which contains an instruction and several demos, each with human-written chain-of-thoughts.

Numerous studies have demonstrated the sensitivity of LLMs to prompts, revealing that minor modifications, such as adding or removing a few tokens or rephrasing the prompt, can significantly impact LLM performance (Liu et al., 2023; Zhu et al., 2023; Jiang et al., 2020). Therefore, prompt design has become a labor-intensive endeavor. Further complicating the matter is the rapid evolution of LLMs, rendering prompts crafted for a specific LLM ineffective when applied to a newer version of LLM. This highlights the need for automatic prompt engineering techniques.

In this paper, we consider the problem of automatic prompt engineering for black-box LLMs, where the tuning algorithms will improve prompts by querying the model. While automatic prompt engineering has been studied recently, existing research (Deng et al., 2022; Xu et al., 2022; Guo et al., 2023; Fernando et al., 2023) focuses on optimizing short instructions with one or a few sentences. These methods either evolve prompts by searching for word replacements or utilize LLMs to rewrite the entire prompt (Xu et al., 2022; Fernando et al., 2023; Guo et al., 2023; Yang et al., 2023). It is challenging to apply them for evolving a long prompt like Figure 1. For long prompts, word replacement-based search faces a large search space, while rewriting the entire prompt using a single LLM query is extremely difficult. Although it is possible to apply these methods to individual sentences, we have observed that tuning a single sentence may not sufficiently improve the perfor-

Findings of the Association for Computational Linguistics: ACL 2024, pages 10672–10685
August 11-16, 2024 ©2024 Association for Computational Linguistics



A prompt found by our algorithm for Disambiguation QA

Clarify the meaning of sentences with ambiguous pronouns.
Q: In the following sentences, explain the antecedent of the pronoun (which thing the pronoun refers to), or state that it is ambiguous.
Sentence: The chief told the counselor that they took the day off.
Options:
(A) The chief took the day off
.....
A: Let's think step by step.
Here we need to determine who the pronoun "they" might be referring to. There are two possible referents for "they", namely the chief and the counselor. The verb "told" might be able to help us determine which one is more likely (if either). ~~Let X be the chief and Y the counselor. The sentence is then of the form "X told Y that (X or Y) did something."~~ **Let X be the chief and Y the advisor. The sentence is of the form "X told Y that (X or Y) did something.**" Let's consider Y first: "X told Y that Y did something." This case does not make much sense, as Y .....
the chief and Y is the counselor, the answer should be the chief. So the answer is (A).
~~Q: In the following sentences, explain the antecedent of the pronoun (which thing the pronoun refers to), or state that it is ambiguous.~~
Q: In the following sentences, identify the antecedent of each pronoun (which thing the pronoun refers to), or state that the antecedent is ambiguous.
.....

Figure 1: **Left panel**: the average accuracy improvements of the proposed method over 3 runs with 50 iterations on the training set. We report the accuracy on the whole set (including training and test sets). More detailed results can be found in Table 2. **Right panel**: An example of a long prompt in BBH (Disambiguation task) which consists of an instruction, several demo examples and chain-of-thought reasoning. The first sentence (Clarify...) is the instruction and each block contains "Q: ... A: ..." is an in-context example. We show that by rewriting a few selected sentences in this long prompt with our proposed method, we can improve the accuracy by more than 10%.

mance of long prompts (see Section 4). Therefore, *how to automatically find better long pompts and what's the potential performance gain by tuning long prompts* remain unanswered in the literature.

In this paper, we propose an algorithm named Automated Prompt Engineering Xpert (APEX). The main operation in APEX is to replace a sentence by a semantically equivalent sentence, which can be done by queries to LLMs. We adopt a greedy algorithm with beam search to make the prompt search more efficient. Further, for long prompt engineering it is important to identify which sentence to change, and to what direction. We thus propose two novel techniques that utilize search history to enhance the effectiveness of LLM-based mutation in our algorithm.

Our contributions can be summarized below:

- This paper presents the first formal discussion of automatic long prompt engineering, demonstrating substantial performance gains on multiple tasks.
- We propose a greedy algorithm with beam search that can rapidly optimize prompts. Furthermore, a novel guided mutation method is introduced to enhance convergence.
- We conducted experiments on the Big Bench Hard (BBH) benchmark, where prompts comprise thousands of tokens, including instructions and chain-of-thought reasoning. Our results demonstrate that the proposed automatic long prompt engineering method significantly enhances performance. Notably, we achieved an average of 9.2% absolute accuracy improvements on 8 tasks selected from BBH, as shown in Figure 1, with only 50 evaluations on the training set. We further demonstrate the effectiveness of APEX on GSM8K across various LLMs.

2 Related Work

Designing effective prompts to fully exploit the capabilities of LLMs has become an important topic. Many principal ways for prompt design have been studied recently (Reynolds and McDonell, 2021; Brown et al., 2020; Wei et al., 2022b; Wang et al., 2022a, 2023a; Jiang et al., 2020). For instance, a well-designed prompt may include a system prompt, an instruction prompt outlining the task, multiple contextual examples, and a chain-of-thoughts reasoning section that explains the step-by-step thought process behind the examples. By incorporating these elements, effective prompts often have tens of sentences and thousands of tokens. Prompt engineering has been used to not only improving the performance of LLMs, but also to achieve certain goals, such as robustness testing (Wallace et al., 2019) and jailbreaking (Zou et al., 2023; Chao et al., 2023).

An orthogonal line of previous work has explored soft-prompt tuning, a technique that op-

timizes prompts within a continuous embedding space using standard continuous optimization algorithms (Lester et al., 2021; Zhang et al., 2021; Wang et al., 2022b). While capable of achieving satisfactory performance, soft-prompts lack interpretability and cannot be applied via LLM APIs. Additionally, these parameter-efficient fine-tuning methods demand large training sets, making them unsuitable for applications with limited data, such as those with only tens or hundreds of samples.

Given the limited availability of training data (e.g., < 1000 samples), our focus lies in exploring strategies for optimizing hard prompts, which are semantically equivalent to the original prompts but yield superior performance. The literature of automatic prompt engineering considers two primary settings. The first setting, which aligns with our work, assumes the existence of an initial human-crafted prompt and aims to refine or improve it to achieve enhanced performance. Several discrete search algorithms have been proposed for this setting: (Shin et al., 2020) searches for the trigger tokens that can improve the performance of language models on downstream tasks. (Xu et al., 2022) employs a genetic algorithm for prompt tuning, utilizing back translation, cloze tasks, and sentence continuation to mutate the initial instruction. (Pryzant et al., 2023) explored the utilization of input-output pair feedback to refine instructions in an iterative manner. More recently, (Fernando et al., 2023; Guo et al., 2023) proposed leveraging LLMs for mutation and crossover operations in evolutionary searches. Although they can also be applied to entire prompt, we show in the main experiment that the proposed algorithm is more effective in evolving an entire prompt. This justifies the proposed ideas (history-guided evolution/sentence selection) can effectively isolate the important part of long prompt to evolve.

In addition, (Yang et al., 2023) demonstrated the optimization capabilities of LLMs in generating improved prompt variations based on previous fitness scores. However, these prompt evolution techniques are designed for short sentences or paragraphs within a long prompt. For example, many of them try to evolve only the instruction part or the sentence "Let's think step by step" in the prompt. Our work aims to provide complete freedom to evolve the entire long prompt, opening up more avenues for improvement but also introducing challenges in determining how and where to change the original prompt.

Other recent concurrent works have also used the idea of LLM as agents to optimize prompt. (Wang et al., 2023b) proposed to model prompt optimization as a state-transition process, where at each state the LLM uses the predicted error to generate feedback to and select next action. This work has a different search space with us, where they aim to correct existing prompts instead of rewriting it. Another work that published after this paper focuses on incorporating human feedback into prompt optimization (Chen et al., 2024).

Another setting focuses on automatic prompt generation without a pre-existing prompt. (Honovich et al., 2022) demonstrated the ability of LLMs to generate brief task descriptions based on input-output pairs. Building upon this technique, (Zhou et al., 2022) proposed an automatic prompt engineering (APE) algorithm capable of generating prompts from given pairs. Under the same problem setting, (Chen et al., 2023) developed a continuous relaxation approach. Beyond these settings, there exists some algorithms developed to automate the generation of chain-of-thought prompts (Zhang et al., 2022). Our work seeks to design a general prompt tuner, which is orthogonal and can be combined with those instruction learning or auto chain-of-thoughts methods.

3 Proposed Method

In this paper, we address the challenge of automatic long prompt engineering for language models. Given a language model $\mathcal{L}$ and an initial prompt p_{init} designed by human for a particular task, our goal is to refine the prompt to achieve superior performance. To enhance the prompt, we are provided with a limited training set (e.g., 100 input-output pairs) $\{(x_i, y_i)\}_{i=1}^{n}$ for performance evaluation. Specifically, for each sample we conduct prediction by $\mathcal{L}([p, x_i])$ and assess its agreement with the corresponding ground truth label y_i. We define score(p) as the performance metric of prompt p on the training set. We consider the scenario where the number of available samples n is limited, which is a common situation where individuals lack sufficient data for model fine-tuning but can still utilize the data to design an improved hard prompt. We will evaluate generalization performance on a hold-out test set that is not used in prompt search.

```
Generate a variation of the following
instruction while keeping the semantic
meaning.

Input: <original sentence>
A variation of the input is:
```

Figure 2: The vanilla LLM-Mutator used in our search.

3.1 Search space

Our goal is to generate a new prompt that is semantically similar to the original prompt while achieving enhanced performance. We avoid introducing non-interpretable tokens, such as adversarial triggers (Zou et al., 2023). The restrictions offer two advantages: 1) the prompts discovered by our algorithm are interpretable, facilitating the verification that the prompt still performs the intended task, and 2) since we are only provided with a limited set of training samples, constraining the search space can mitigate the issue of overfitting.

To conduct prompt search, we decompose the long prompt into m individual sentences:

$$p = [s_{(1)}, s_{(2)}, \ldots, s_{(m)}].$$

We then allow each sentence to be rephrased while preserving its semantic meaning. Since LLMs excel at sentence rephrasing, we adopt LLM to generate semantically equivalent variation of a given sentence and call it **LLM-Mutator**. The prompt we used for vanilla LLM mutator can be found in Figure 2. An improved version of LLM-Mutator will be introduced later in Section 3.3.

3.2 Search algorithm

A straightforward approach to conduct search using the LLM-Mutator is the following greedy algorithm. At each iteration, we randomly select a sentence, denoted by $s_{(i)}$, and utilize the LLM-Mutator to generate an alternative sentence $s'_{(i)}$. We then replace $s_{(i)}$ by $s'_{(i)}$ in the old prompt p if the new resulting prompt improves the performance. However, we observed that this vanilla greedy approach can be easily trapped in local optima. In our problem, the training set is so limited that sometimes a detrimental modification is not reflected in the training score. As a result, unfavorable edit are sometimes accepted due to insufficient evaluation, which hurts the prompt's future improvement.

To address this issue, we propose conducting a beam search by maintaining a pool of k top-performing prompts, denoted as $p_1^*, p_2^*, \ldots, p_k^*$. At each iteration, we randomly select one of these k prompts and refine the chosen prompt. We then evaluate this new candidate and maintain the top-k prompt pool. This approach ensures that even with the introduction of detrimental edits, recovery from such errors is still possible as we retain not only the top prompt. Our experiments reveal that this approach leads to significantly improved training and test performance compared to the pure greedy algorithm, as demonstrated in Figure 3. Further discussions of the implementation details and comparison between beam search and greedy methods are in Appendix 7.1.

It is worth noting that our method is closely related to the Genetic Algorithm (GA). GA is a widely recognized method for discrete optimization involving black-box functions. Assuming $P = \{p_1, \ldots, p_k\}$ represents the solutions in the current pool, GA applies mutation and crossover operations to this pool to generate a set of newly proposed candidates $P' = \{p'_1, \ldots, p'_k\}$. The fitness scores (performance of the solutions) of these newly proposed candidates are then evaluated. Subsequently, only the top k solutions in $P \cup P'$ are retained before proceeding to the next iteration.

While GA possesses a high exploration capability, it is slower in improving training accuracy in the first 50 iterations. However, given the vastness of our search space and the reasonably good quality of the human-written initial prompt, a majority of mutations and crossovers performed in the first few iterations yield poorly performing candidates. Evaluating each of these candidates is computationally expensive, even though it can enhance solution diversity.

Our algorithm is effectively a "greedy" version of GA. We maintain a candidate set of size k and update the candidate pool immediately upon generating each new candidate, rather than waiting for the evaluation of the entire generation of offspring before updating the pool. As a result, our solution pool remains more up-to-date, leading to faster convergence compared to GA. Figure 3 further verifies this observation, demonstrating the slower initial growth of GA's learning curve.

3.3 History-guided search

Randomly rephrasing a sentence in a prompt is akin to conducting a random mutation within the space of semantically equivalent sentences, which is not very efficient. This subsection introduces our **main innovation**, where we show how the search history can be employed to guide the mutation in a more purposeful direction.

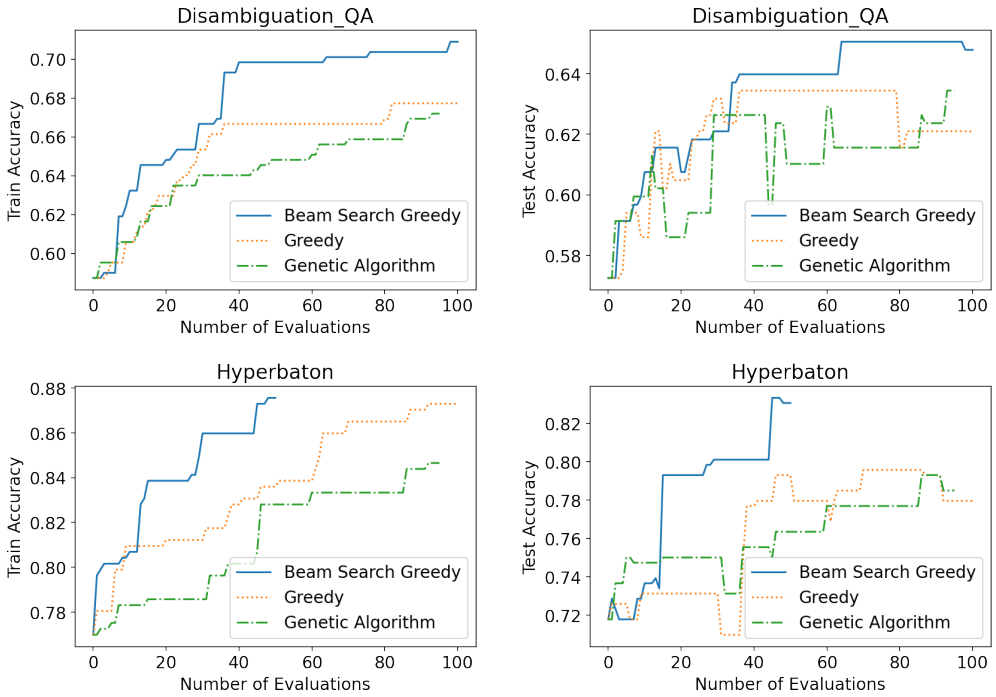


Figure 3: Comparison between beam-search greedy, greedy, and genetic algorithm. This is the average results over three runs. Beam-search greedy outperforms both greedy algorithm and genetic algorithm.

Guided mutation for a single sentence. At iteration T, we can denote the search history as $\{(s_t^{\text{before}}, s_t^{\text{after}}, r_t)\}_{t=1}^{T-1}$, where s_t^{before} is the sentence before mutation, s_t^{after} is the sentence after mutation, and the reward r_t indicates the change in the score (r_t is positive if the mutation enhances the performance, and negative otherwise). Assume s_T is the current sentence that is selected for mutation, we can then use the history to guide the mutation towards more positive reward. For example, if we know rephrasing the sentence "So, it is true that Lesley is a great-grandfather of Leroy" to "Therefore Leslie is Leroy's great-grandfather." can improve the performance, then for another sentence "So, it is true that everyone who is an ancestor of Dana is a stepbrother of Brian" we may want to rewrite it as "Therefore everyone that is an ancestor of Dana is Brian's stepbrother."

It has been shown that LLMs are able to learn from in-context examples (Zhang et al., 2021). Therefore, we propose to let the LLM-Mutator in-context learn from the history. This can be done by listing history in the prompt: if $r_t > 0$, we include an in-context example $s_t^{\text{before}} \Rightarrow s_t^{\text{after}}$, and if $r_t < 0$ we include $s_t^{\text{after}} \Rightarrow s_t^{\text{before}}$. The prompt for LLM-Evolver is shown in Figure 4.

Since the history can be long and it has been shown that including too many in-context examples can be harmful, we only retrieve a small set of

```
You are a professional sentence rephraser. Your job is
to take a given sentence, and produce a new sentence
that is easier for language models to understand.

Here are some examples of how rephrasing a sentence can
improve the performance of a language model:
<original> s1 </original>
<rephrased> s1' </rephrased>
.....
Rephrase the next sentence to enhance the language
model's performance.
<original> sT </original>
<rephrased>
```

Figure 4: The guided LLM-mutator used in our search.

relevant history entries when rephrasing a sentence. Specifically, when evolving s_T, we compute the similarity between s_T and each s_t^{before}, which is calculated by $\phi(s_T)^T \phi(s_t^{\text{before}})$ and ϕ is a sentence encoder (we use a T5 (Raffel et al., 2020) in our experiments). We then only select entries that pass a certain threshold to include as in-context examples.

Guided sampling for sentence selection. Long prompt typically contains tens or hundreds of sentences, posting a challenge to the search algorithm. Our experiments demonstrate that altering only a few sentences in a long prompt can significantly enhance its effectiveness, but the changes need to be at right places. It is thus beneficial to bias the sampling distribution towards selecting more impactful sentences for modification.

We model sentence selection as a contextual bandit problem (Langford and Zhang, 2007). In contex-

tual bandit, at each iteration a learner is faced with m arms associante with feature vectors $x_1, \ldots, x_m$, and the learner is trying to pull an arm at each round to optimize the overall reward. In our case, m arms correspond to the m sentences $s_{(1)}, \ldots, s_{(m)}$ in a prompt, and features can be obtained by $\phi(s_{(i)})$ where ϕ is a text encoder. Utilizing feature information is crucial because if modifying a particular sentence can lead to performance improvement, it is likely that modifying similar sentences will also yield positive results.

We then adopt the Lin-UCB algorithm (Li et al., 2010) to guide sentence selection. Given the history $\{(s_t^{\text{before}}, s_t^{\text{after}}, r_t)\}_{t=1}^{T-1}$, we compute a linear estimator of the underlying reward model

$$w_T^* = (H^T H + \lambda I)^{-1} H^T r, \tag{1}$$

where H is a $(T-1)$-by-d matrix with each row being $\phi(s_t^{\text{before}})$, and $r = [r_1, \ldots, r_{T-1}]^T$. This is simply a solution of a ridge regression with feature matrix H and reward r. The following UCB rule is then used to select the sentence for mutation:

$$\arg\max_i \phi(s_{(i)})^T w_T^* + \alpha\sqrt{\phi(s_{(i)})^T A^{-1} \phi(s_{(i)})}, \tag{2}$$

where $A = H^T H + \lambda I$. Since this Lin-UCB estimation may not be accurate, we let the algorithm has probability P to choose a purely random arm and $1 - P$ to choose the sentence based on Eq. (2). The algorithm is summarized in Algorithm 1.

Algorithm 1 Automated Prompt Engineering Xpert (APEX)

p_0: initial prompt; k: beam size; θ: threshold.
Initialize solution pool P as $\{p_0\}$
for iter $= 1, 2, \ldots$ **do**
  Randomly select p from top-k solutions in P
  $p \leftarrow$ random number in $[0, 1]$
  if $p < 0.5$ **then**
    Sample a random sentence $s_{(i)}$
    Choose $s_{(i)}$ by Eq. (2).
  end if
  Compute $T = \{t : \phi(S_{(i)})^T \phi(s_t^{\text{before}}) > \theta\}$
  $s'_{(i)} \leftarrow$ Guided Mutation on $s_{(i)}$ with history T
  Evaluate $p' = [s_{(1)}, \ldots, s_{i-1}, s'_i, s_{i+1}, \ldots]$
  Add p' to P
end for

4 Experimental Results

In this section, we present empirical evidence demonstrating that APEX can significantly enhance performance on the Big-Bench Hard (BBH) benchmark (Suzgun et al., 2022; Srivastava et al., 2022). We then conduct ablation studies and test the performance on GSM8K dataset across various LLM models. Finally we show qualitative results on how APEX refines human-written prompts.

4.1 Experimental Settings

We consider the prompt developed in Suzgun et al. (2022) for the BBH tasks, where prompts consist of two parts: Task Description and Demos. The Task Description provides instructions describing the task, while each demo includes the question, a chain-of-thoughts demonstration illustrating the problem-solving process step-by-step, and the final answer. During prompt tuning, the format (e.g., "Question. ", "Answer. ") are retained, while the tuning algorithm is allowed to modify any other parts including the instruction part and the chain-of-thoughts part. Table 1 summarizes the statistics of the datasets used in this study. The full prompts for each data can be downloaded online[1].

Each task consists of 250 samples, randomly divided into 50% for training and 50% for testing. Only training samples are utilized for prompt tuning. All tasks are multi-class classification problems, and we report accuracy on training data, test data, and the combined dataset when comparing different tuning methods.

For most of the experiments, we evaluated the performance on text-bison model and utilized the instruction-tuned PaLM 2-L model as LLM mutator. Both models belong to the PaLM 2-model family (Anil et al., 2023). We also demonstrate that our method can be applied to other LLMs, where we conduct experiments on GPT-3.5 in Section 4.3.1. The GPT version used in our experiment is gpt-3.5-turbo-instruct-0914. Following the previous prompt tuning works, we set temperature as 0. To enhance the diversity of sentence mutation, we set the temperature of the LLM-Mutator to 0.5.

We compare the proposed algorithm with the following baselines:

- Original Prompt: Performance of the original human-designed prompt developed in (Suzgun et al., 2022), which also serves as the

[1] https://github.com/suzgunmirac/BIG-Bench-Hard

initialization for other tuning methods.

- Genetic Algorithm: An implementation of the genetic algorithm for long prompt tuning. To facilitate a more comparable comparison with our method, we set the pool size to 4. At each step, 8 new candidates are generated by randomly mutating and performing crossover on the top candidates within the pool.
- Evolve "step-by-step": Several recent prompt tuning studies have explored the concept of evolving a prompt using a single sentence, such as the 'Let's think step by step' sentence employed in chain-of-thought prompts (Kojima et al., 2022). We adopted one of the state-of-the-art methods for single sentence optimization (Yang et al., 2023) to optimize all of the 'Let's think step by step' sentences within the chain-of-thought prompt.
- APO (Pryzant et al., 2023) designed a prompt optimization algorithm which expands the instruction space by a feedback loop. We apply this algorithm to directly tune long prompts.
- PromptBreeder (Fernando et al., 2023) uses a genetic algorithm to search for prompts. We incorporate LLM-based mutation and crossover without optimizing the meta prompt.
- Greedy: A simple greedy approach mentioned in Section 3.2, where we only store the top performing candidate in the pool and generating new candidate on top of it.

For our method (APEX), we use the same hyperparameters for all the experiments. We set the pool size (beam size) to 4 for all experiments. When applying guided mutation in Section 3.3, we use a T5 encoder to encode both the current entry and the history and normalize the embeddings to have a unit ℓ_2 norm. We retrieve only the top 4 history entries and require the ℓ_2 distance between the encoded history and the current sentence to be below 0.5. When applying the sentence selection algorithm described in Section 3.3, we set $\alpha = 0.05$ in Eq. (2) and set $P = 0.5$. Note that those hyperparameters are not well tuned.

4.2 Main Results: Results on Big Bench Hard

For this experiment, we limited the computational budget to 50 evaluations on the training set and reported the training, test, and combined accuracy achieved by each method. All the experiments are run three times and we report the mean and standard deviation. The results are summarized in Table 2. APEX outperform all other methods, demonstrating significant improvements in accuracy across all tasks. We also compute the improvements of accuracy on the whole dataset (including both training and testing) and report the accuracy gain in Figure 1.

Dataset	Number of Words	Number of Mutable Sentences
Causal Judgement	678	20
Salient Translation	745	34
Disambiguation	671	37
Formal Fallacies	741	32
Hyperbaton	539	19
Dyck Language	679	22
Color Reasoning	435	17
Logical Five	461	25

Table 1: Initial prompt statistics for each dataset. When calculating the "number of mutable sentences," we only consider sentences that can be modified during the search phase.

Across all 8 tasks, APEX achieves an average of 8.2% gain in test accuracy and 9.2% gain in the accuracy of full evaluation set (train + test). Among these tasks, we achieve the largest performance gain (18.45%) on the logical deduction task, and the smallest gain (2.45%) on reasoning about colored objects (Color Reasoning). One potential reason of the marginal gain on color reasoning is that the original prompt already achieves high accuracy, so there is not much room for improvements.

Comparing the baselines, it becomes evident that evolving a single sentence (Evolve 'step-by-step') fails to achieve substantial improvements in long prompt tuning. This is mainly due to the fact that long prompt tuning typically involves over 20 sentences with detailed instructions and explanations, so naively modifying "Let's think step by step" is unlikely to lead to significant enhancements. Furthermore, the Genetic Algorithm and Greedy Algorithm each exhibit their own limitations, as discussed in Section 3.2.

On the other hand, we show that conducting APO over the long prompt leads to suboptimal results, often lower than GA and Greedy (baselines for long prompt tuning). This is mainly due to the difficulty of rewriting the whole long prompt. For example, the original prompt of Hyperbaton contains 19 sentences with more than 500 words, but most of the prompt generated by APO has only <50 words. As a result, it couldn't find a better prompt. This explains why evolving the whole long prompt together can fail.

Despite being able to significantly boost the per-

Task	Original Prompt		Genetic Algorithm		Evolve "Step-by-step"		Greedy		APO		PromptBreeder		APEX	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Causal Judgment	58.1	59.6	63.1	59.2	63.1	58.2	63.0	59.6	64.5	60.6	65.2	63.1	67.7	**63.7**
Salient Translation	54.4	54.4	58.7	56.5	57.1	55.7	59.5	53.6	55.2	52.8	57.1	50.3	60.0	**58.7**
Disambiguation	58.7	57.3	64.8	61.0	64.6	61.8	66.7	63.4	68.3	65.3	70.8	**71.5**	70.1	68.0
Formal Fallacies	58.9	61.1	63.5	64.5	60.6	62.4	68.5	68.8	65.8	68.4	68.8	64.8	70.4	**73.1**
Hyperbaton	77.0	71.8	82.8	76.3	84.9	79.6	83.5	79.3	77.0	71.8	79.6	80.1	88.4	**85.5**
Dyck Language	15.1	17.0	20.2	16.4	18.25	20.4	19.0	17.3	22.4	19.8	21.9	**22.7**	22.8	20.9
Color Reasoning	79.4	82.3	85.1	78.9	86.2	81.5	85.1	81.1	81.0	75.0	82.4	80.7	85.4	**83.5**
Logical Five	33.7	40.7	49.3	48.2	54.5	50.3	53.9	51.3	53.2	46.8	51.3	48.9	59.8	**54.7**
Average	54.4	55.5	60.94	57.6	61.2	58.7	62.4	59.3	60.9	57.5	62.1	60.3	65.6	**63.5**

Table 2: Main results on BBH benchmark. The search budget is limited to 50 evaluations over the training set for each method. We ran each experiment 3 times and report the mean and standard deviation. Due to the space constraint and to ensure readability, the standard deviations are deferred to Table 10.

formance, we also observe some degree of overfitting in our search procedure. Although the proposed beam search method can partially mitigate overfitting (see Section 3.2), we still observe higher training accuracy than testing in most cases. However, as the performance gain is substantial, the prompts found by the algorithm are still significantly better on the test set.

Can previous methods handle long prompts? Although many previous methods can be directly applied to long prompts, our empirical finding in Table 2 demonstrated that they cannot achieve good performance by rewriting the entire prompt. In particular, the APO and PromptBreeder algorithms in our comparison directly rewrote the entire prompt and achieve lower performance than APEX. Another option is to use existing work and tune only a particular sentence. In Table 2, we show that tuning the "step-by-step" sentence by (Yang et al., 2023) is insufficient, and further, we conduct another experiment in Table 3 showing that only tuning the instruction part of the prompt is insufficient.

Can we directly use training samples for in-context learning? In our method and all the baseline prompt tuning algorithms, the training samples are used for obtaining a better prompt. However, another way to utilize training samples is to add all of them as demos in the in-context learning framework. To compare with this approach, we show the results of using all training samples in In-context Learning (ICL) and show the result in Table 3. We can see adding all samples in ICL improves the performance in some tasks (especially in Color Reasoning), but could also degrade the performances. Overall, APEX is still more effective in most of the cases comparing with ICL with all training samples.

4.3 Ablation Study

We study APEX under different LLMs, datasets, and by varying the components in APEX.

4.3.1 Different LLMs and datasets

To show the performance of APEX under different settings such as different datasets and LLMs, we conduct experiments on the GSM8K dataset (Cobbe et al., 2021) using different LLMs. We choose the "prompt_hardest" prompt[2] as the initial prompt which contains 103 lines with 1,639 words. We conduct prompt tuning with 1,000 sampled training data for our method and evaluate the revised prompt on the full test set. For text-bison and Palm-2-L we use Palm-2-L as the mutation LLM, while for GPT-3.5 we use GPT-3.5 as the mutation LLM. The results are shown in Table 5. We observe that the proposed method consistently improves the performance with different LLMs including text-bison, GPT-3.5-instruct and Palm 2-L.

4.3.2 Different components of the algorithm

We conduct an ablation study on the two techniques introduced in Section 3.3: the history-guided mutation and the contextual bandit algorithm for sentence selection. To verify whether both techniques are essential in APEX, we consider two settings: APEX without history-guided mutation (during mutation, we randomly rewrite a sentence using the prompt in Figure 2) and APEX without sentence selection (randomly select a sentence to mutate, but using guided mutation Figure 4 at each iteration). All the hyperparameters are fixed in this experiment. The results are presented in Table 4. We can observe that both components are contributing to the final performance of the model.

[2] https://github.com/FranxYao/chain-of-thought-hub

Task	Causal	Translation	Disambiguation	Fallacies	Hyperbaton	Dyck	Color	Logical
Original	59.6	54.4	57.3	61.1	71.8	17	82.3	40.7
APO (instruction only)	61.7	50.2	59.7	71.8	76.3	23.7	60.7	44.1
Evolve "step-by-step"	58.2	55.7	61.8	62.4	79.6	20.4	81.5	50.3
ICL Prompt (all samples)	61.8	52.0	**69.7**	63.7	75	**25**	68.9	41.1
APEX (whole prompt)	**63.7**	**68.7**	68.0	**73.1**	**85.5**	20.9	**83.5**	**54.7**

Table 3: Comparing APEX (tuning the whole prompt) with other methods when tuning a smaller part of the prompt.

Task	Disambiguation		Formal Fallacies		Hyperbaton		Logical Five	
	Train	Test	Train	Test	Train	Test	Train	Test
APEX (no history-guided mutation)	67.4	66.9	61.1	66.1	83.6	78.1	55.6	53.2
APEX (no sentence selection)	68.2	63.1	71.2	72.8	81.4	80.3	54.2	50.4
APEX	70.1	68.0	70.4	73.1	88.4	85.5	59.8	54.7

Table 4: Ablation Study on the two components introduced in Section 3.3.

Model	Original Prompt	APEX
text-bison	0.657	0.705
GPT-3.5-instruct	0.765	0.801
Palm 2-L	0.841	0.872

Table 5: GSM8K results on three LLMs.

4.4 Qualitative results

One important benefit of automatic hard prompt engineering is that the resulting prompts remain interpretable by humans, allowing users to easily verify the modifications. We provide some successful examples found by our search in Table 6 and Table 7 (appendix). In each table, we show only the different parts of the initial prompt from BBH and the changes made by our method.

The first example demonstrated in Table 6 (appendix) is for the logic deduction task on five objects. The initial prompt achieves 38.8% accuracy while the revised prompt found at iteration 48 improves the performance to 57.9% train accuracy and 54.0% test accuracy. We observed that most of the changes involve minor revisions to the original sentence without altering its meaning. These seemingly insignificant modifications can lead to substantial improvements in LLM accuracy, showcasing the important of automatic long prompt engineering. We also include another example in Appendix 7.2.

5 Conclusions

We study the problem of automatic prompt engineering for long prompts, often comprising thousands of tokens. We investigate the performance of the standard greedy algorithm and genetic algorithm, and develop a search algorithm that yields superior performance. With only 50 evaluations on the training set, our method achieves an average absolute accuracy improvement of 9.2% across 8 tasks from Big Bench Hard. This demonstrates the significant potential benefits of automatic long prompt tuning and underscores the importance of this emerging area.

6 Acknowledgements

We thank Ruochen Wang, Vineet Gupta, Kedar Dhamdhere, Daliang Li for valuable discussions and feedback.

Limitations

As the first paper focusing on automatic engineering of the entire long prompts, we identify several limitations of the current methods which can lead to interesting future research:

- The current algorithm relies on using another LLM to rephrase a sentence. As illustrated in Section 4.4, this LLM-mutator may introduce errors during sentence rewriting, particularly for intricate sentences (e.g., CoT in complicated logical deduction tasks). Therefore, improving the "correctness" of LLM-Mutator is an interesting future area of research, which has not been fully addressed in our work as well as other recent studies (Fernando et al., 2023; Guo et al., 2023).
- In the current implementation, we break down the long prompt into individual sentences and modify one sentence at a time. However, it might be beneficial to manipulate multiple sentences simultaneously during mutation or consolidate multiple sentences into a single one. An automated mechanism for carrying out this process would be an interesting direction for enhancing our method.
- Although our algorithm is able to find a good solution with less than 100 evaluations on the training set, the cost is still not negligible especially when tuning prompts using APIs with

cost or rate limits. Employing early stopping techniques, where the evaluation of poorly performing candidates are terminated early, could potentially reduce the number of queries.

- Although automatic prompt engineering can achieve significant gain, the search space of hard prompt has limited representation power which hinders further performance improvements. It has been shown that soft prompt tuning has limited representation power (Wang et al., 2023c), and our search space is a small subset of soft prompts. Therefore, when provided with sufficient data, computational resources, and white-box access to the LLM, (parameter-efficient) fine-tuning may still achieve superior performance.

References

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

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.

Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J Pappas, and Eric Wong. 2023. Jailbreaking black box large language models in twenty queries. *arXiv preprint arXiv:2310.08419*.

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

Yongchao Chen, Jacob Arkin, Yilun Hao, Yang Zhang, Nicholas Roy, and Chuchu Fan. 2024. Prompt optimization in multi-step tasks (promst): Integrating human feedback and preference alignment. *arXiv preprint arXiv:2402.08702*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

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*.

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

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*.

Or Honovich, Uri Shaham, Samuel R Bowman, and Omer Levy. 2022. Instruction induction: From few examples to natural language task descriptions. *arXiv preprint arXiv:2205.10782*.

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.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

John Langford and Tong Zhang. 2007. The epoch-greedy algorithm for contextual multi-armed bandits. *Advances in neural information processing systems*, 20(1):96–1.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Lihong Li, Wei Chu, John Langford, and Robert E Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35.

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.

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*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–7.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. *arXiv preprint arXiv:2010.15980*.

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

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

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019. Universal adversarial triggers for attacking and analyzing nlp. *arXiv preprint arXiv:1908.07125*.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023a. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*.

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

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

Yihan Wang, Jatin Chauhan, Wei Wang, and Cho-Jui Hsieh. 2023c. Universality and limitations of prompt tuning. *arXiv preprint arXiv:2305.18787*.

Yihan Wang, Si Si, Daliang Li, Michal Lukasik, Felix Yu, Cho-Jui Hsieh, Inderjit S Dhillon, and Sanjiv Kumar. 2022b. Preserving in-context learning ability in large language model fine-tuning. *arXiv preprint arXiv:2211.00635*.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022a. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022b. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

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*.

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

Ningyu Zhang, Luoqiu Li, Xiang Chen, Shumin Deng, Zhen Bi, Chuanqi Tan, Fei Huang, and Huajun Chen. 2021. Differentiable prompt makes pre-trained language models better few-shot learners. *arXiv preprint arXiv:2108.13161*.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2022. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*.

Kaijie Zhu, Jindong Wang, Jiaheng Zhou, Zichen Wang, Hao Chen, Yidong Wang, Linyi Yang, Wei Ye, Neil Zhenqiang Gong, Yue Zhang, et al. 2023. Promptbench: Towards evaluating the robustness of large language models on adversarial prompts. *arXiv preprint arXiv:2306.04528*.

Andy Zou, Zifan Wang, J Zico Kolter, and Matt Fredrikson. 2023. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*.

7 Appendix

7.1 Comparison between greedy, beam search and GA.

Here we include more implementation details of greedy v.s. beam search algorithm for reproducing Figure 3, and discuss why beam search performs significantly better.

In the implementation of greedy algorithm, we maintain the top candidate (based on training accuracy), and randomly mutate a sentence of the top candidate at each iteration. We then evaluate the new prompt on training data. If the new prompt achieves better training accuracy than the previous top prompt, we replace the top prompt with this new prompt. Otherwise we keep the original top prompt. The process is simple and there is no hyper-parameter. On the other hand, the beam search algorithm maintains a beam size of k. At each iteration, it randomly selects a prompt from the beam, and randomly mutates one sentence to generate a new prompt. We then evaluate the prompt, and decide whether we need to update the candidates in the beam. Therefore, the only hyper-parameter is k and we set it to 4 in all our experiments. We found the algorithm is not very sensitive to k, as long as it is not too large.

We observe greedy search is very easy to saturate (accuracy not improving anymore). The main difference between greedy search and beam search is that the greedy search always through away all other candidates except the one with top training accuracy. However, the search will stuck when this top candidate has good training accuracy but bad generalization accuracy (so it achieves good training accuracy by "luck" and is actually a bad prompt). In this case, all the later prompts in the greedy algorithm are generated from this "bad" candidate which leads to bad performance, and it is hard to improve anymore. In contrast, the beam search algorithm always has a chance to recover from those "bad" candidates, since there are 4 candidates in the pool and we always have chance to derive later prompt from another candidate in the beam.

We also include more details for reproducing Figure 3. As described above, Greedy doesn't have any hyper-parmaeter. Beam search has a hyper-parameter k (beam size) which is set to 4 for all experiments in this paper. In order to make Genetic Algorithm (GA) comparable with beam search (see more details in Section 3.2), we maintain a pool of size 4 for genetic algorithm. At each iteration, we randomly generate 4 candidates by randomly mutating from the pool, and another 4 candidates by cross-over. We have tried to further increase these numbers, and the GA will have even slower progress in the initial phase. For example, if we have 8 mutations and 8 cross-overs, then 1 iteration will require 16 evaluations (and it can be observe that beam search already achieves a high performance after 40-50 evaluations).

7.2 Additional Qualitative Result

The second prompt in Table 7 is for the Formal Fallacies task, designed to identify whether a given logic statement is valid. The initial prompt achieves 60% accuracy while the revised prompt found at iteration 91 improves the train accuracy to 92.1% and test accuracy to 83.1%. Similar to the previous case, most of the changes involve minor revisions. In this case, we also want to highlight a potential limitation of the proposed method that could be addressed in future work. In the sentence marked as *, the revised sentence is *not* semantically equivalent to the original one. The original and revised sentences represent different logical statements (original sentence: not A $\rightarrow$ B; new sentence: B $\rightarrow$ not A). However, the LLM appears incapable of detecting this subtle distinction, leading to an erroneous rephrasing. Although this change leads to improve training accuracy, it actually hurts test accuracy. Therefore, incorrect mutations can lead to overfitting, and developing strategies to mitigate these errors during the search process would be an interesting area of future research.

7.3 Experimental results with longer runs

In Table 2 we have demonstrated the results with 50 evaluations. Here we further show the performance of our algorithms versus several baselines when running with 100 iterations. The results are shown in Table 9. We can observe that APEX still outperforms other methods with 100 iterations.

7.4 Ablation study on in-context example selection

In this subsection, we conduct an experiment to demonstrate that having too many in-context examples is harmful for guided mutation, which justifies the proposed history selection algorithm. We compare the original APEX with two variants: APEX (random history) which has the same number of

Original prompt: 38.8% accuracy	New prompt found at iteration 48: 56% accuracy (train 57.9% / test 54.0%)
A: Let's think step by step.	A: Let's think things through one step at a time.
(2) Eli finished below Amy: "(above) ? Amy ? Eli ? (below)".	(2) Amy was above Eli: "(above) ? Amy ? Eli ? (below)".
Eli finished last.	Eli came in last place.
Q: The following paragraphs each describe a set of three objects arranged in a fixed order. The statements are logically consistent within each paragraph. On a shelf, there are three books: a white book, a green book, and an orange book. The green book is to the right of the white book. The orange book is the rightmost.	Q: Each paragraph below describes three objects arranged in a fixed order, and the statements are logically consistent in each paragraph. There are three books on a shelf: a white, green, and orange book. The green book is to the right of the white book, and the orange book is in the far right position.
A: Let's think step by step.	A: Let's think through things one step at a time.
The white book is the leftmost.	The white book is at the far left.
A: Let's think step by step.	A: Let's think one step at a time.

Table 6: An example that APEX improves the performance on Logical Deduction (Five) from 38.8% to 56%.

Original prompt: 60% accuracy	New prompt found at iteration 91: 76% accuracy (train 92.1%, test 83.1%)
Distinguish deductively valid arguments from formal fallacies.	Identify deductively valid arguments from formal fallacies.
So, it is true that Lesley is a great-grandfather of Leroy.	So it's true that Lesley is Leroy's great-grandfather.
Whoever is not a great-grandfather of Clyde is a stepbrother of Brian: If X = NOT (great-grandfather(Clyde)), then X = stepbrother(Brian).	* If someone is a stepbrother of Brian then they are not a great-grandfather of Clyde.
Furthermore, by (1), we have if X = NOT (great-grandfather(Clyde)), then X = stepbrother(Brian).	Additionally, according to (1) we have that if X = NOT (great-grandfather(Clyde)), then X = stepbrother(Brian).
By the transitive relation rule in first-order logic, we then have: if X = ancestor(Dana), then X = stepbrother(Brian).	Using the transitive relation rule in first-order logic, we have: if X = ancestor(Dana), then X = stepbrother(Brian).
Let's see whether the Hypothesis can be deduced from the arguments (1) and (2) by logical reasoning?	Let's see whether the Hypothesis is a logical consequence of the arguments (1) and (2)?
So, from (1) and (2), we cannot necessarily deduce the Hypothesis.	So, given (1) and (2), we are not always entitled to infer the Hypothesis.

Table 7: An example demonstrating how our method changes the original prompt. By conducting these changes the combined accuracy on Formal Fallacies can be improved from 60% to 76%. Interestingly, there's one line

Task	Disambiguation		Formal Fallacies	
	Train	Test	Train	Test
APEX (random 20 history)	65.4	63.7	62.3	65.7
APEX (random history)	68.7	66.5	69.8	72.4
APEX	70.1	68.0	70.4	73.1

Table 8: Ablation Study on the history selection approaches in APEX. We observe retrieval-based selection leads to some performance improvements, and selecting too many in-context examples will hurt mutation.

in-context examples as APEX but select the examples randomly, and APEX (random 20 history) which randomly selects 20 in-context examples for guided mutation. The results in Table 8 demonstrates that (1) retrieval based sample selection as proposed in Section 3.3 leads to 1-2% improvements, and (2) selecting too many examples will hurt the performance significantly.

Task	Disambiguation		Formal Fallacies		Hyperbaton		Logical Five	
	Train	Test	Train	Test	Train	Test	Train	Test
Original Prompt	58.7	57.3	58.9	61.1	77.0	71.8	33.7	40.7
Greedy (50 iters)	66.7±0.6	63.4±2.7	68.5±2.6	68.8±3.0	83.5±3.3	79.3±5.3	53.9±3.9	51.3±1.2
Greedy (100 iters)	67.7±1.0	62.1±2.0	68.5±2.6	68.8±3.0	86.9±1.9	78.0±5.9	58.7±3.2	54.5±3.5
GA (50 iters)	64.8±1.6	61.0±0.7	63.5±1.1	64.5±0.7	82.8±1.3	76.3±2.5	49.3±2.7	48.2±1.6
GA (100 iters)	67.2±0.4	63.4±1.7	64.1±0.9	67.4±1.4	84.7±0.4	78.5±0.8	55.7±0.4	52.8±2.0
APEX (50 iters)	70.1±1.9	68.0±2.1	70.4±3.9	73.1±3.6	88.4±2.6	85.5±1.1	59.8±2.2	54.7±0.9
APEX (100 iters)	72.4±1.7	69.3±2.3	70.9 ± 2.5	74.4±4.2	89.2±3.1	86.3±2.4	60.6±2.5	55.7±2.0

Table 9: Comparison of Apex, GA and Greedy with 100 iterations. Results show that APEX outperforms other methods with 100 iterations.

Task	Original Prompt		Genetic Algorithm		Evolve "Step-by-step"		Greedy		APEX	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Causal Judgment	58.1	59.6	63.1±1.3	59.2±0.5	63.1±0.5	58.2±0.5	63.0±1.0	59.6±2.9	67.7±1.5	63.7±1.3
Salient Translation	54.4	54.4	58.7±2.0	56.5±4.3	57.1±0.4	55.7±0.4	59.5±1.5	53.6±3.3	60.0±3.5	58.7±2.6
Disambiguation	58.7	57.3	64.8±1.6	61.0±0.7	64.6±1.5	61.8±1.0	66.7±0.6	63.4±2.7	70.1±1.9	68.0±2.1
Formal Fallacies	58.9	61.1	63.5±1.1	64.5±0.7	60.6±1.0	62.4±1.7	68.5±2.6	68.8±3.0	70.4±3.9	73.1±3.6
Hyperbaton	77.0	71.8	82.8±1.3	76.3±2.5	84.9±0.0	79.6±0.3	83.5±3.3	79.3±5.3	88.4±2.6	85.5±1.1
Dyck Language	15.1	17.0	20.2±1.7	16.4±0.4	18.25±0.0	20.4±0.1	19.0±0.4	17.3±0.6	22.8±0.7	20.9±0.4
Color Reasoning	79.4	82.3	85.1±0.4	78.9±1.7	86.2±0.7	81.5±1.3	85.1±0.7	81.1±0.4	85.4±0.4	83.5±0.4
Logical Five	33.7	40.7	49.3±2.7	48.2±1.6	54.5±1.0	50.3±0.4	53.9±3.9	51.3±1.2	59.8±2.2	54.7±0.9
Average	54.4	55.5	60.94	57.6	61.2	58.7	62.4	59.3	65.6	63.5

Table 10: Main results on BBH benchmark. The search budget is limited to 50 evaluations over the training set for each method. We ran each experiment 3 times and report the mean and standard deviation. This is the same results as table 2 with standard deviations reported.