Genetic Prompt Search via Exploiting Language Model Probabilities

Jiangjiang Zhao\textsuperscript{1,2}, Zhuoran Wang\textsuperscript{3}, Fangchun Yang\textsuperscript{1}

\textsuperscript{1}Beijing University of Posts and Telecommunications, P.R. China
\textsuperscript{2}China Mobile Online Services Co., Ltd. Beijing, P.R. China
\textsuperscript{3}Clouchie Limited, London, United Kingdom

zjjbupt@bupt.edu.cn, wangzhuoran@clouchie.ai, fcyang@bupt.edu.cn

Abstract
Prompt tuning for large-scale pretrained language models (PLMs) has shown remarkable potential, especially in low-resource scenarios such as few-shot learning. Moreover, derivative-free optimisation (DFO) techniques make it possible to tune prompts for a black-box PLM to better fit downstream tasks. However, there are usually preconditions to apply existing DFO-based prompt tuning methods, e.g. the backbone PLM needs to provide extra APIs so that hidden states (and/or embedding vectors) can be injected into it as continuous prompts, or carefully designed (discrete) manual prompts need to be available beforehand, serving as the initial states of the tuning algorithm. To waive such preconditions and make DFO-based prompt tuning ready for general use, this paper introduces a novel genetic algorithm (GA) that evolves from empty prompts, and uses the predictive probabilities derived from the backbone PLM(s) on the basis of a (few-shot) training set to guide the token selection process during prompt mutations. Experimental results on diverse benchmark datasets show that the proposed precondition-free method significantly outperforms the existing DFO-style counterparts that require preconditions, including black-box tuning, genetic prompt search and gradient-free instructional prompt search.

1 Introduction
The recent successes of pretrained language models (PLMs) are revolutionising the field of natural language processing (NLP) [Devlin et al., 2019; Radford et al., 2019; Liu et al., 2019; Raffel et al., 2020; Clark et al., 2020]. Meanwhile, extremely large PLMs have demonstrated great potential in few-shot learning scenarios (e.g. [Brown et al., 2020]), which makes them increasingly attractive as out-of-box tools for general use. Fine-tuning such large-scale PLMs can still be computationally expensive, even on a few-shot training set. But this is significantly relieved by a new paradigm called prompt tuning. Prompt tuning methods [Li and Liang, 2021; Gao et al., 2021; Lester et al., 2021; Shin et al., 2020; Liu et al., 2022; Liu et al., 2021a; Liu et al., 2023] work by inserting a small number of tunable variables into a backbone PLM’s input (and sometimes also the hidden states [Liu et al., 2021a]) to bias its predictive probabilities towards desired output, while keeping the backbone model’s parameters frozen during the learning process. The tunable variables here can either be continuous vectors (namely soft prompts) or surface tokens (discrete prompts), for which obtaining promising values is the essential objective and can be solved by either gradient-based optimisers [Kingma and Ba, 2015; Loshchilov and Hutter, 2019] or derivative-free optimisation (DFO) techniques [Kolda et al., 2003; Rios and Sahinidis, 2013; Yu and Gen, 2010].

To democratise the access to those very large PLMs, a common practice is to deliver them as back-box services with cloud APIs only [Brown et al., 2020; Ouyang et al., 2022] (a.k.a LM-as-a-Service [Sun et al., 2022b]). This implies prompt tuning via DFO to be an senseful and important research direction, where the backbone’s parameters are not required to be exposed to the tuner. Existing work has proven the feasibility of applying DFO for prompt tuning [Xu et al., 2022; Prasad et al., 2022; Sun et al., 2022b; Sun et al., 2022a]. For example, Xu et al. [2022] and Prasad et al. [2022] introduced search heuristics to refine human generated (discrete) prompts via edit operations such as paraphrase, cloze, deletion, swap, etc. On the other hand, Sun et al. [2022b; 2022a] proposed the so-call ‘black-box tuning (BBT)’ methods for continuous prompt optimisation based on the co-variance matrix adaptation evolution strategy [Hansen et al., 2003].

However, the limitations of existing methods are obvious. The discrete prompt search heuristics in [Xu et al., 2022; Prasad et al., 2022] require carefully designed manual prompts available beforehand, which yield additional human efforts. Furthermore, their performance may also highly rely on the suitableness of those initial manual prompts (cf. §4.3). In addition, methods of this kind usually require auxiliary language models (LMs) to paraphrase existing prompts (e.g. T5\textsubscript{11B} [Raffel et al., 2020] used in [Xu et al., 2022] and PEGASUS [Zhang et al., 2020] used in [Prasad et al., 2022]), which involves extra dependencies. For BBT-style continuous prompt optimisation [Sun et al., 2022b; Sun et al., 2022a], it requires the backbone PLM to provide extra APIs so that the continuous prompts can be injected as word embeddings (or hidden states for BBTv2 [Sun et
can not only be attainable in practice. A na"ıve alternative is in-context learning (ICL) proposed along with GPT-3 [Brown et al., 2020], which simply prepends training examples to the input as the prompt. But prepending too many or too long examples could make the final input exceed the sequence length threshold of the backbone model, which restricts ICL's applicable tasks. In addition, the performance of ICL is usually less competitive, as seen in [Sun et al., 2022b; Sun et al., 2022a; Xu et al., 2022; Gao et al., 2021] and also in our experiments (cf. §4.3).

This paper aims to waive the above preconditions required by the existing DFO-based prompt tuning methods, and presents a novel genetic algorithm (GA) [Mitchell, 1998] that generates discrete prompts from the ground up. The proposed method, named Generic Algorithm for Predictive Probability guided Prompting (GAP3)1, works as follows. Firstly, discontinuous prompt chunks are considered as chromosomes, with prompt tokens being genes. Then, starting from an empty one, GAP3 evolves the prompts via chromosome crossovers and gene mutations. At each mutation step, either a new mask token is inserted into a random chromosome at a random position, or a random existing gene is masked. After this, the masked slot will be filled by a token that approximately maximises the predictive probability of the ground-truth labels on a (few-shot) training set. The algorithm iterates for a predefined number of steps, with individuals consisting of diverse chromosomes/genes competing to survive and breed, according to their fitness scores computed on the training set.

In comparison to existing DFO-based prompting methods, the major advantages of the proposed GAP3 are three-fold:

1. No extra APIs for vector injections are required, since GAP3 searches for discrete prompts.

2. No manual prompts are required, as GAP3 is initialised with an empty prompt.

3. For backbone PLMs that yield predictive probabilities for masked tokens, e.g. masked language models (MLMs) [Devlin et al., 2019; Liu et al., 2019] or T5-style encoder-decoder networks [Raffel et al., 2020; Lewis et al., 2020], GAP3 generates prompt tokens directly based on the backbone itself, without needing an auxiliary LM, which further reduces preconditions for its applications. (If a casual LM [Radford et al., 2018; Radford et al., 2019] is the backbone of interest, GAP3 will need an auxiliary MLM for token generation (cf. §4.3), where the MLM can also be a black-box model.)

Taking RoBERTaLARGE [Liu et al., 2019] and GPT-2LARGE [Radford et al., 2019] as the backbone PLMs of interest, respectively, the performance of GAP3 is evaluated on 7 benchmark datasets (the same as those used in [Sun et al., 2022b]), in comparison with that of the existing counterparts, including BBT [Sun et al., 2022b], gradient-free intrinsic prompt search (GPS) [Xu et al., 2022], gradient-free instructional prompt search (GRIPS) [Prasad et al., 2022], as well as ICL [Brown et al., 2020]. The experimental results prove the effectiveness of the proposed GAP3, where it outperforms all the DFO-style baselines, achieving at least 2.9% and 2.4% absolute improvements in the average scores for RoBERTaLARGE and GPT-2LARGE, respectively.

2 Related Work

Parameter-efficient tuning (PET). PET reduces the cost of adapting a large PLM to downstream tasks, by tuning only a small proportion of the parameters instead of the full model [Houlsby et al., 2019; Pfeiffer et al., 2020]. Prompt tuning [Lester et al., 2021; Liu et al., 2022; Liu et al., 2021a; Liu et al., 2021b; Qin and Eisner, 2021] form a sub-direction of PET, where the tunable parameters are the injected soft prompts. Despite the success of PET methods, they are not suitable for the growing trend of LM-as-a-Service deployments, as gradient-based optimisations are required.

Discrete prompt search. Discrete prompts are more preferable in black-box scenarios, since no model-level modifications are involved. Manually created intuitive prompts were found helpful in earlier research [Petroni et al., 2019; Schick and Schütze, 2021b; Schick and Schütze, 2021a], but they are suboptimal in a general sense. For methods that searches for prompts automatically, paraphrasing is a commonly used methodology to expand the existing prompt set (usually initialised with manual prompts) for succeeding search heuristics to winnow [Xu et al., 2022; Prasad et al., 2022]. In addition, Hou et al. [2022] proposed to ensemble prompts via boosting. But such ensemble multiplies the inference cost at the same time. Reinforcement learning (RL) has recently been employed by Deng et al. [2022] and Diao et al. [2023] to optimise discrete prompts for black-box backbones. It’s worth noting that the method proposed in Deng et al. [2022] involves action explorations in a considerably large prompt space, which results in significantly more API calls to train the model than its counterparts (such as BBT [Sun et al., 2022b]). Other proposed methodologies include mining prompt templates from the web [Jiang et al., 2021] and training specific prompt generators [Ben-David et al., 2022], which correspond to extra computational and human efforts.

Hybrid methods. It is also possible to tune a backbone model based on both discrete prompts and differentiable parameters, of which typical examples include AutoPrompt [Shin et al., 2020] who search for discrete prompt tokens based on gradients, and LM-BFF [Gao et al., 2021] that combines automatic prompt generation and model fine-tuning. Methods of this kind should be considered as more comparable to PET, as they violate the black-box assumption.

3 Methodology

3.1 Prompt Template

Assume that the downstream task is to classify the input text [X] to a label [Y]. A prompt template here means a permutation to arrange [X], [Y] and the prompt [T], e.g. template ‘[X] [T] [Y]’ stands for placing the prompt between the input text and the label. We further assume that a task’s input may consist of multiple text pieces (e.g. when classifying a pair of sentences) and prompt chunks are allowed

1Code and supplementary material available at: https://github.com/zjihit/gap3
to be placed at multiple positions. Therefore, we generalise the above template representation with subscripts, e.g. \([X_1 \{T_1\} |X_2 \{T_2\} \{Y\}]\) means two prompt chunks inserted between two text pieces and between the second text piece and the label, respectively.

### 3.2 Genetic Algorithm for Prompting

For a given downstream task and a training set, the proposed GAP3 works as follows. Firstly, we define a prompt template (cf. §3.1). The template here only decides a permutation of the text pieces. The actually prompt chunks are initialised as empty sets at this stage. To interpret the process in GA-style, we regard each prompt chunk \([T_i]\) as a chromosome, where the actual tokens to be filled into it are regarded as genes. In addition, we call entire prompts with diverse chromosomes individuals. After this, we generate the first population of \(N\) individuals by randomly mutate one gene of the initial empty individual, where \(N\) is a predefined hyperparameter. Concretely, in this very first step, the mutation means randomly selecting a chromosome (that is blank), and inserting a token to it. Then the algorithm iterates as follows.

1. Evaluate each individual on the training set, to obtain a fitness score;
2. Keep top \(\sqrt{N}\) most fit individuals, called elites;
3. Randomly draw pairs of individuals from the elites to perform (probabilistic) chromosome crossovers, until a new population of size \(N\) is yielded.
4. Mutate each new individual’s gene (probabilistically) by randomly inserting a new token or replacing a random existing token with a new one;

The above process will be repeated for \(M\) steps before the best individual being chosen as the final output, where \(M\) is also a predefined hyperparameter.

**Elite selection.** Besides ranking the current population of individuals according to their fitness scores, we also maintain the overall best individual achieved in the previous iterations. If the ‘best so far’ individual surpasses the current elites, we will add it into the current elite group (while dropping the tail elite). This is to avoid good genes being lost during the crossovers and mutations.

**Crossover.** Elite pairs are drawn in a weighted roulette wheel manner, with respect to their fitness scores. After this, for each \([T_i]\), with probability \(\rho_c\), we exchange the corresponding chromosomes between an elite pair, to yield two new individuals. The hyperparameter \(\rho_c\) is called the crossover probability.

**Mutation.** With probability \(\rho_m\) (called the mutation probability), we mutate an individual. When this happens, we randomly draw an action between ‘insert’ or ‘replace’ with even probabilities. If the ‘insert’ action is applied, a mask token will be inserted at a random position of a random chromosome. Otherwise, a random existing gene (of a random chromosome) will be masked (by replacing that token with a mask token). This implies, in either way, there will be one mask token in the current prompt. Then, \(LM\) probabilities will be exploited to realise that mask to a real token.

### Algorithm 1 GAP3

**Input:**
- \(D\) – training set
- \(T_0\) – initial (empty) template
- \(LM\) – backbone model

**Hyperparameter:**
- \(M\) – number of iterations
- \(N\) – population size
- \(\rho_c\) – crossover probability
- \(\rho_m\) – mutation probability

**Output:** \(T^*\) – the most fit prompt

1. \(G \leftarrow \{\}\), \(n \leftarrow \text{int}(\sqrt{N})\), \(T^* \leftarrow \text{NULL}\)
2. for \(1 \ldots N\) do
3. \(T \leftarrow T_0\)
4. \(T \leftarrow \text{mutate}(T,D,LM,\text{prob}=1.0)\)
5. \(G \leftarrow G \cup \{T\}\)
6. end for
7. for \(1 \ldots M\) do
8. \(G \leftarrow \text{sort}(G,\text{key=fitness}(G,D,LM))\)
9. if \(\text{better than}(G[0],T^*)\) then
10. \(T^* \leftarrow G[0], E \leftarrow G[0:n]\)
11. else
12. \(E \leftarrow \{T^*\} \cup G[0:n-1]\)
13. end if
14. \(G' \leftarrow \{\}\)
15. while \(|G'| < N\) do
16. \(T, T' \leftarrow \text{roulette}(E,\text{weights=fitness}(E,D,LM))\)
17. \(T, T' \leftarrow \text{crossover}(T,T',\text{prob}=\rho_c)\)
18. \(T \leftarrow \text{mutate}(T,D,LM,\text{prob}=\rho_m)\)
19. \(T' \leftarrow \text{mutate}(T',D,LM,\text{prob}=\rho_m)\)
20. \(G' \leftarrow G' \cup \{T,T'\}\)
21. end while
22. \(G \leftarrow G'\)
23. end for
24. return \(T^*\)

Algorithm 1 gives the pseudo-code of the proposed GAP3, where hyperparameters and constant objects are denoted in italic type. We leave the detailed explanations of the token selection process and the consequential design of the fitness function to §3.3 and §3.4, respectively, to keep the discussions on the main algorithm coherent here.

### 3.3 Mutation Guided by LM Probabilities

**Notations.** Let \((x, y)\) denote a data example, where \(x\) is the input token sequence, and \(y\) is the label (token). Without loss of generality, for tasks involving \(m\) pieces of text as input, we will let \(x := (x_1, \ldots, x_m)\), with each \(x_i\) consisting of a token sequence. Then a prompt \(T\) can be regarded as a function that applies the prompt tokens to the data point \((x, y)\), according to its associated template, to obtain a final token sequence, as \(T(x, y)\). We use \(t_i\) to denote the token indexed by \(i\) in \(T\), and let \(T^*_i\) stand for the prompt with \(t_i\) replaced by a mask token. We also use \(T^*_i\) to refer the \(i\)-indexed masked token. Similarly, \((x, y^n)\) stands for the data point its label \(y\) masked.

Firstly, assume we have an ‘ideal’ \(LM\) that satisfies the Bayes’ rule. Then, for an arbitrary data point and an arbi-
We have:

\[
P(\hat{y} = y | \mathcal{D}, \mathcal{T}, t) = \frac{P(t^* = t | \mathcal{T}, \hat{y} = y)}{P(t^* = t | \mathcal{T}, \hat{y} = y^*)}
\]

which yields:

\[
P(\hat{y} = y | \mathcal{D}, \mathcal{T}, t) = \frac{P(t^* = t | \mathcal{T}, \hat{y} = y)}{P(t^* = t | \mathcal{T}, \hat{y} = y^*)}
\]

where \(P(\cdot | \cdot)\) stands for the conditional probability given by the LM, and \(\mathcal{T}, t\) stands for the prompt obtained by substituting the masked token \(t^*\) with token \(t\). We call Eq. 1 an ‘ideal’ assumption, because it holds if and only if the probabilities here are ground truth probabilities, which will not be achievable in practice (since general PLMs are not trained subject to such a constraint to satisfy the Bayes’ rule).

Therefore, to make the equation valid, we introduce a bias item \(\lambda\) and reformulate Eq. 2 in logarithmic form, as:

\[
\log P(\hat{y} = y | \mathcal{D}, \mathcal{T}, t) = \log P(t^* = t | \mathcal{T}, \hat{y} = y) - \lambda(x, y, t, \mathcal{T}, t)
\]

where we name the probabilities in the form of \(P(\cdot | \cdot)\), for the ease of reference in future discussions. Now, recall the mutation process in §3.2. Given an arbitrary prompt \(t^*\) with a masked token, and a training set \(\mathcal{D}\), one would want to unmask \(\mathcal{T}\) by seeking the token \(t\) that maximising the label posterior on \(\mathcal{D}\), as:

\[
i = \arg \max_{t \in \mathcal{V}} \sum_{(x, y) \in \mathcal{D}} \log P(y | \mathcal{D}, \mathcal{T}, t) = \arg \max_{t \in \mathcal{V}} \log P(\hat{y} = y | \mathcal{D}, \mathcal{T}, t)
\]

where \(\mathcal{V}\) is the vocabulary of the LM. If we omit the biases \((\lambda)\), \(P(y | \mathcal{D})\) can be computed tractably based on right-hand side of Eq. 3, which implies invoking the LM twice for each training example \((x, y)\), by feeding it with \(\mathcal{T}\) and \(\mathcal{T}\) respectively. However, the biases \((\lambda)\) here are indispensable, while computing them for all possible data-prompt-token-mask combinations is obviously intractable. Hence, we design a heuristic to address this, as follows.

Let \(\lambda(t) := \max_{(x, y, t, \mathcal{T}, t)} \lambda(x, y, t, \mathcal{T}, t)\). If we replace \(\lambda(x, y, t, \mathcal{T}, t)\) in Eq. 3 with \(\lambda(t)\), the left-hand side becomes a lower bound of the original \(P(y | \mathcal{D})\). Firstly, we initialise \(\lambda(t)\) as 0 for all \(t \in \mathcal{V}\). We define \(P(y | \mathcal{D}) := \log P(t^* = t | \mathcal{T}, \hat{y} = y) + \log P(y | \mathcal{D}) - \lambda(t)\) (represented in simplified notations). Then, at each time a prompt is mutated, we perform a two-step update, as:

\[
i \leftarrow \arg \max_{t \in \mathcal{V}} \sum_{(x, y) \in \mathcal{D}} \log P(y | \mathcal{D})
\]

\[
\lambda(t) \leftarrow \max_{t \in \mathcal{V}} \left[ \lambda(t), \max_{D} \left\{ \log P(y | \mathcal{D}) - \log P(y | \mathcal{D}) \right\} \right]
\]

where \(\log P(y | \mathcal{D})\) is also computed based on the LM. Note here, \(P(y | \mathcal{D})\) essentially evaluates the performance of the obtained prompt on the training data, as is also used by the fitness function (cf. §3.4). Therefore, computing it is an inevitable effort, instead of an extra cost.

**Remark on \(\lambda\).** A more intuitive explanation of \(\lambda(t)\)’s function is that, it penalises those tokens that tend to occur repeatedly but will overestimate the predictive probabilities of the labels.

**Remark on Eq. 5.** In practice, always adopting the top-1 token \(t\) may yield duplicated prompts (especially when generating the initial population (cf. §3.2)). Therefore, we actually collect top-\(n\) tokens based on \(\sum_{D} \log P(y | \mathcal{D})\). Then, starting from the first one, we examine whether the prompt unmasked based on the current token has already been seen previously. If yes, we move to the next token, until an unseen consequent prompt is obtained.

**Masked LM vs. casual LM.** The most elegant part of the proposed GAP3 is that, if the backbone PLM is an MLM (or a T5-like encoder-decoder network (Raffel et al., 2020; Lewis et al., 2020)), all the four predictive probabilities \(P(t | y), P(y | t), P(t | y), P(t | y)\) can be obtained from the backbone model itself. Nevertheless, if the backbone is a casual LM who can only predict \(P(y | t)\), we can use an auxiliary MLM to compute \(P(t | y), P(y | t)\) and \(P(t | y)\), in which case, \(\lambda(t)\) to a great extent prevents the algorithm repeatedly generating tokens highly biased to the auxiliary.

### 3.4 Fitness Function

For a given task, the actual objective metric (such as accuracy or F1-score) on the training set will be a straightforward measure of the fitness for those individuals yielded in the GA. However, in the few-shot learning case, it will be very easy to have many individuals achieving a same metric score. Too many indistinguishable individuals occurring in a population may result in less chance of breeding given to those potentially more competitive genes.

Therefore, in GAP3, we actually make the fitness measure two-dimensional. The task-specific objective metric is the dominant fitness. If (and only if) two individuals have an equal score in the dominant fitness, we further compare them according to a secondary fitness score. Too many indistinguishable individuals occurring in a population may result in less chance of breeding given to those potentially more competitive genes.

Therefore, in GAP3, we actually make the fitness measure two-dimensional. The task-specific objective metric is the dominant fitness. If (and only if) two individuals have an equal score in the dominant fitness, we further compare them according to a secondary fitness. The secondary fitness score is computed as:

\[
F_{2nd}(T) = \frac{1}{|D|} \sum_{(x, y) \in D} \left[ \delta_{y, y} \frac{P(x, y | \mathcal{T}(x, y))}{\sum_{y' \in \mathcal{Y}} P(x, y') \mathcal{D}'} \right]
\]

\[
\hat{y} = \arg \max_{y' \in \mathcal{Y}} P(y' | \mathcal{T}(x, y))
\]

where \(\mathcal{Y}\) denotes the task’s label set, \(\delta_{y, y}\) is the Kronecker delta function. \(F_{2nd}\) means that we renormalise the predictive probabilities on the label set, and average over the ‘hinge’ probabilities, where incorrectly predicted examples contribute zero values. Note here, in the weighted roulette wheel selection process, we only use \(F_{2nd}\) scores for the weights, as they are more distinguishable and partially reflects the classification accuracy.
4 Experiments

We conduct a group of main comparative experiments and an ablation study. §4.1, §4.2 and §4.3 describe the settings, baselines and results for the main experiments, respectively. Those for the ablation study is presented in §4.4, specifically.

4.1 Settings

Datasets. The datasets used in the main experiments consist of 7 benchmark NLP tasks, which are the same as in [Sun et al., 2022b], including Yelp polarity, AG’s News and DBPe-dia from [Zhang et al., 2015], SST-2, MRPC and RTE from the GLUE benchmarks [Wang et al., 2018], as well as SNLI [Bowman et al., 2015]. The experiments are in a k-shot learning setting, where for each task, we randomly sample $k = 16$ examples for each label from the original training set. For SST-2, MRPC and RTE, we use their development sets as the test sets. For the other tasks, we use the original test sets. For MRPC, $F_1$ is used as the evaluation metric, while accuracy is the metric for all the other tasks.

Backbone PLMs. We choose RoBERTa$_{LARGE}$ [Liu et al., 2019] and GPT-2$_{LARGE}$ [Radford et al., 2019] as backbones, conducting two types of experiments, respectively. For the GPT-2$_{LARGE}$ backbone, we use RoBERTa$_{LARGE}$ and BERT$_{BASE}$ (cased) [Devlin et al., 2019], respectively, as the auxiliaries for GAP3 (cf. §3.3).

Hyperparameters for GAP3. We set GAP3’s population size $N = 64$ and iteration number $M = 50$, with crossover and mutation probabilities $\rho_c = 0.5$ and $\rho_m = 0.75$, respectively. For the RoBERTa$_{LARGE}$ backbone, the above settings result in $M \times N \times (1 + 2\rho_m) = 8000$ expected number of API calls (one call for fitness plus two calls for mutation (with probability $\rho_m$) per individual per data example). For the GPT-2$_{LARGE}$ backbone, the same settings correspond to 3200 API calls to GPT-2$_{LARGE}$ and 4800 to RoBERTa$_{LARGE}$/BERT$_{BASE}$. The prompt templates and label words for GAP3 on each task can be found in appendix A.

4.2 Baselines

We choose the following existing DFO-based prompting methods as baselines. The implementations of the baselines are all based on the original source code provided by their authors.

BBT. BBT [Sun et al., 2022b] requires for an additional k-shot development set, which is also randomly sampled from the original training sets of the tasks, without overlapping with the k-shot training examples. BBT’s budget for API calls is set to 8000 (the same as in [Sun et al., 2022b]), with the prompt length 50.

GPS. GPS [Xu et al., 2022] is another GA-based prompting method that evolves by using $T_{51B}$ [Raffel et al., 2020] to paraphrase the prompts. As GPS requires multiple manual prompts to initialise the first population, we only conduct experiments for it on Yelp polarity, AG’s News, SNLI and RTE. We use the manual prompts presented in [Schick and Schütze, 2022] to initialise the Yelp polarity and AG’s News experiments. Experiments on SNLI are initialised with the ANLI manual prompts presented in [Sanh et al., 2022]. Sanh et al. [2022] also provides a manual prompt set for RTE, which is directly adopted here. We set GPS’ population size to 25 and number of iterations to 10.

GRIPS. GRIPS [Prasad et al., 2022] is also a heuristic search based prompting method, which evolve prompts based on PEGASUS [Zhang et al., 2020] and generic token-level edits. GRIPS requires manual instructions to initialise. For the GPT-2$_{LARGE}$ experiments, we initialise it based on the Natural-Instructions dataset [Mishra et al., 2022], and set the iteration number to 50 with 100 candidates generated per iteration. However, based on the same settings, we failed to obtain any reasonable results for the RoBERTa$_{LARGE}$ backbone. GRIPS either fails to find a valid update or yields results no better than chance. (This may be because either GRIPS itself or the initial instructional manual prompt is unsuitable for an MLM backbone.) Therefore, we omit the comparison with GRIPS for the RoBERTa$_{LARGE}$ backbone.

In addition, we also compare our GAP3’s performance with that of ICL [Brown et al., 2020], manual prompts, gradient-based prompt tuning (PT) [Li and Liang, 2021] as well as full-model fine-tuning (FT). For ICL, we concatenate the k-shot training examples in a random order (but with a balanced label distribution) to form a prefix prompt for the input. In regard to manual prompts, for Yelp polarity, AG’s News, SNLI and RTE, we test all the available prompts and choose the best score. For the other tasks, we just use the simple prompt templates given in [Sun et al., 2022b]. For PT and full-model FT, Adam optimisers [Kingma and Ba, 2015] are employed. For PT, with learning rate 5e-4 and batch size 16, it runs for 1000 epochs. For full-model FT, with the same batch size, but learning rate 1e-5, we run it for 200 epochs.

We did not choose BBTv2 [Sun et al., 2022a] as a baseline in this work, because we consider hidden-state injection as a much stronger violation to the black-box assumption, which will be unfair to other methods.

Label words. We use label words slightly different from those in [Sun et al., 2022b]. Because in our case, we intuitively expect the label words more substitutable to each other from natural language point of view. In order to eliminate the bias in the experimental results caused by this difference, for BBT and ICL, we experiment them with both the label words in [Sun et al., 2022b] and ours, and choose the better results obtained. For GPS and GRIPS, the label words are embedded in their initial manual prompts, which we keep unchanged.

Fairness of resources used. For the RoBERTa$_{LARGE}$ backbone, our GAP3 and BBT are compared under the same API call budget. However, BBT uses an additional k-shot development set, which means doubling the number of labelled examples required. If we assume the general cost of an API call to be linear to the scale of the model behind, for the GPT-2$_{LARGE}$ backbone, GAP3 would be much more cost-efficient than BBT, as 60% of its API calls are spent on the auxiliary RoBERTa$_{LARGE}$/BERT$_{BASE}$ that is much smaller than GPT-2$_{LARGE}$. Based on the same assumption, we can consider the resources used by GAP3, GPS and GRIPS as approximately comparable.
4.3 Results

Experimental results with RoBERTa\textsubscript{LARGE} and GPT-2\textsubscript{LARGE} as the backbones are shown in Table 1 and 2, respectively. It can be found that in both scenarios, the proposed GAP3 outperforms the other baselines in the gradient-free group with a notable margin. Interestingly, for the GPT-2\textsubscript{LARGE} backbone, the BERT\textsubscript{BASE} (110M) auxiliary works almost as good as the RoBERTa\textsubscript{LARGE} (354M) auxiliary. Although showing a slightly lower average score, the former achieves even higher scores on SST-2, DBPedia and MRPC than the latter. This also suggests that GAP3’s dependence on a particular auxiliary MLM is weak. In addition, GAP3 surpasses gradient-based PT for RoBERTa\textsubscript{LARGE}, and achieves an average score close to gradient-based PT for GPT-2\textsubscript{LARGE}.

However, full-model FT still appears to be the most competitive paradigm. Despite the capability to tune black-box backbones, none of the methods in the gradient-free group achieves an overall score comparable to full-model FT. Similar findings were also indicated in a recent study [Chen et al., 2022] particularly designed to analyse this aspect. In addition, on RTE, none of the gradient-free methods performs notably better than chance, which indicates the existence of particular problems that are more difficult for DFO-style algorithms to solve.

4.4 Ablation Study

Ablation experiments are conducted based on the SST-2 and AG’s News datasets and the RoBERTa\textsubscript{LARGE} backbone, where we vary one hyperparameter, while keeping the others fixed. The default hyperparameter values are the same as those in §4.1, except that we use 32-shot learning by default in this section, to reduce the variance over 3 different runs.

\textbf{\textit{k}-shot.} We increase the number of training examples per label (i.e. \textit{k}), with \textit{k} being 16, 32, 64 and 128, respectively,
and plot the corresponding performance of GAP3 in Figure 1. It can be found that the obtained accuracy scores grow with \( k \) on both dataset sets. Furthermore, up to \( k = 128 \), there is no clear trend of convergence appearing, which suggests that GAP3 has the potential capability of learning from larger training sets.

**API call budget.** The expected number of API calls for GAP3 is jointly determined by three hyperparameters, the population size \( (N) \), the number of iterations \( (M) \) and the mutation probability \( (\rho_m) \). We plot the ablation results of the above hyperparameters together in Figure 2, against the number of API calls they yield. It is understandable that more API calls normally correspond to better results. However, due to the randomness in GAP3’s strategies, exceptions may occur by chance, where a setting with fewer API calls happen to outperform those with more API calls.

**Other hyperparameters.** As shown in Figure 3, GAP3 is to some extent sensitive to the crossover probability. It suggests that some further heuristics would need to be designed in the future, to seek an optimal value for \( \rho_c \). In addition, we also experiment with an alternative secondary fitness function (cf. §3.4), which is obtained by omitting the Kronecker delta in Eq. 7. The results indicate that doing so will reduce the mean accuracy by 1.0% on SST-2 and 0.5% on AG’s News.

**5 Further Discussions**

**Label word selection.** The label words in this work are manually assigned. Preliminary attempts were made to search for label words automatically, based on the method proposed in [Gao et al., 2021], which, however, resulted in serious overfitting. Better strategies to gain label words for GAP3 will be addressed in our feature research.

**Prompt length.** In GAP3, prompt length is not predefined, but the iteration number hyperparameter will determine an upper threshold of the maximum possible prompt length. In addition, the template defined in §3.1 does not necessarily mean that every prompt slot \([T_i]\) will have tokens in the end. There are possibilities that the final survival individual has some of its chromosomes remain empty. The above features make the prompts generated by GAP3 more flexible and less hyperparameter-dependent.

**Interpretability of prompts.** Generally speaking, the prompts generated by GAP3 are not understandable by human, since it is not designed to gain human-readable text. Nevertheless, one can still find some ‘keywords’ within the prompts interpretable. Example prompts learned in §4.3 can be found in the supplementary material\(^2\).

Limitation. Due to the $M \times N \times (1+2\rho_m)$ API calls yielded during GAP3’s evolutions, it will be computationally expensive to apply it directly to a full-sized training set. This is also a common limitation of the existing DFO-based prompting methods. In our GAP3 case, reshaping the evolution and evaluation processes in a $k$-fold manner will possibly relieve the computational complexity problem, which we will further investigate in future studies.

6 Conclusion
This paper introduces GAP3, an LM probability guided GA, to search prompts automatically for black-box PLM backbones. Despite its outstanding performance on diverse benchmarks, the most significant superiority of GAP3 is the waiver of the preconditions required by existing DFO-based prompting methods, such as injection APIs or manual prompts. The zero or minimal dependency of GAP3 on additional resources suggests it to be an out-of-box complementary to those LM-as-a-Service instances. The computational cost of applying it to full-sized training problems would be the current major limitation of GAP3. Addressing this limitation will be one of our future research directions.

A Prompt Templates and Labels for GAP3
Prompt templates and label words (verbalisers) used in our GAP3 experiments for the RoBERTa\textsubscript{LARGE} and GPT-2\textsubscript{LARGE} backbones are listed in Table 3 and 4, respectively.

<table>
<thead>
<tr>
<th>Task</th>
<th>Template</th>
<th>Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>SST-2</td>
<td>$[T_1] [X] [T_2] [Y]$</td>
<td>bad, good</td>
</tr>
<tr>
<td>Yelp P.</td>
<td>$[T_1] [X] [T_2] [Y]$</td>
<td>bad, good</td>
</tr>
<tr>
<td>AG’s News</td>
<td>$[X] [T] [Y]$</td>
<td>world, sports, business, technology</td>
</tr>
<tr>
<td></td>
<td></td>
<td>company, education, artist, athlete, office, transportation, building, nature, village, animal, plant, album, film, literature</td>
</tr>
<tr>
<td>DBPedia</td>
<td>$[T_1] [Y] [T_2] [X]$</td>
<td>–</td>
</tr>
<tr>
<td>MRPC</td>
<td>$[T_1] [X_1] [Y] [T_2] [X_2]$</td>
<td>No, Yes</td>
</tr>
<tr>
<td>SNLI</td>
<td>$[T_1] [X_1] [Y] [T_2] [X_2]$</td>
<td>Yes, Maybe, No</td>
</tr>
<tr>
<td>RTE</td>
<td>$[T_1] [X_1] [Y] [T_2] [X_2]$</td>
<td>Yes, No</td>
</tr>
</tbody>
</table>

Table 3: Templates and labels for the RoBERTa\textsubscript{LARGE} backbone.

<table>
<thead>
<tr>
<th>Task</th>
<th>Template</th>
<th>Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>SST-2</td>
<td>$[T_1] [X] [T_2] [Y]$</td>
<td>–</td>
</tr>
<tr>
<td>Yelp P.</td>
<td>$[T_1] [X] [T_2] [Y]$</td>
<td>–</td>
</tr>
<tr>
<td>AG’s News</td>
<td>$[X] [T] [Y]$</td>
<td>–</td>
</tr>
<tr>
<td>DBPedia</td>
<td>$[T_1] [X] [T_2] [Y]$</td>
<td>–</td>
</tr>
<tr>
<td>MRPC</td>
<td>$[T_1] [X_1] [T_2] [X_2] [T_3] [Y]$</td>
<td>no, yes</td>
</tr>
<tr>
<td>SNLI</td>
<td>$[T_1] [X_1] [T_2] [X_2] [T_3] [Y]$</td>
<td>always, sometimes, never</td>
</tr>
<tr>
<td>RTE</td>
<td>$[T_1] [X_1] [T_2] [X_2] [T_3] [Y]$</td>
<td>true, false</td>
</tr>
</tbody>
</table>

Table 4: Templates and labels for the GPT-2\textsubscript{LARGE} backbone. ‘–’ stands for labels identical to those in Table 3.

Acknowledgements
The first author thanks the members of the Algorithm and Development Team at the Data Intelligence Centre of China Mobile Online Services Co., Ltd. for fruitful discussions on this work. The second author thanks XREAL for supporting this research by providing the computing power.
References


[Liu et al., 2022] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-Tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of ACL (Volume 2: Short Papers)*, 2022.


