Balancing Multiple Objectives for Efficient Metaprompts for Data Labeling Tasks with Extensive Guidelines

Tobias Schnabel & Jennifer Neville
Microsoft Research
Redmond, WA
{toschnab,jeneville}@microsoft.com

Abstract

Spurred by ever increasing context-window sizes, two recent trends in the application of large language models (LLMs) for data annotation and pattern extraction are (i) longer prompts with complex structures, rich information and task instructions and (ii) the processing of many data points in the same prompt (minibatching) to increase query efficiency. In the process of annotating and analyzing data, the same *metaprompts* are re-used with many different inputs and are thus worth being optimized for length as billing is proportional to overall token usage.

Traditional prompt optimization techniques are only insufficiently addressing those two trends: First, by ignoring the structure of prompts, they are limited in the transformation operations they can perform and second, they do not consider important factors such as input and output costs or adherence to output specifications. To overcome these limitations, we propose structure-aware multi-objective metaprompt optimization (SAMMO), a framework that automatically balances multiple objectives for high level prompt structures and encompasses several existing prompt optimization methods as special cases.

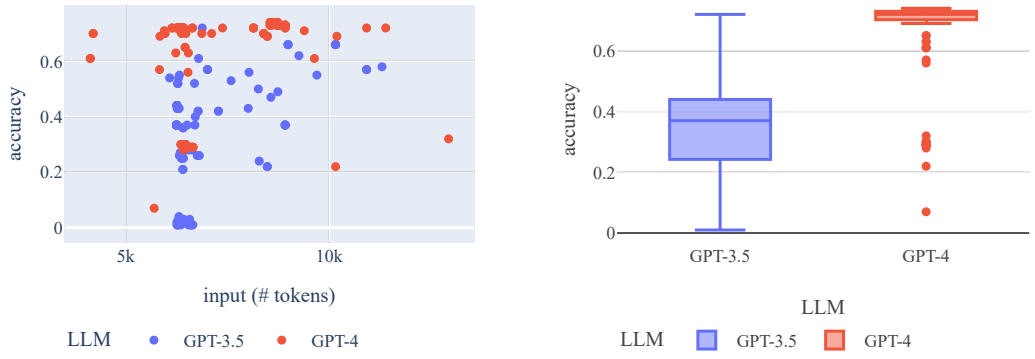
Drawing from approaches for neural architecture search, SAMMO carries out a genetic search over a set of mutation operators that can change the structure and information contained in non-trivial ways. Empirically, we show on a wide range of annotation tasks that SAMMO succeeds in finding metaprompts that have over 30% fewer tokens while still as accurate as the baseline prompt.

1 Introduction

With the recent advent of LLMs that allow for long inputs and outputs, such as ChatGPT and GPT-4, models can be given much richer context and more detailed instructions. While some of the data labeling tasks can rely on pre-defined categories such semantic analysis, many of the more valuable insights require data being labeled with a custom set of complex categories and guidelines. To feed this information to LLMs, practioners have to design a *metaprompt* that defines the task, labeling guidelines, provides examples and describes the input and output format.

In this work, we consider the problem of using a metaprompt π for labeling task $\mathcal{T}$, which takes user input X and labels it with annotation Y, i.e., $\text{LLM}(\pi[X]) = Y$. Conceptually, a *metaprompt* can be thought of as a shared prompt template that is re-used with many different inputs X. We consider longer metaprompts with more complex structure/instructions that are likely to be used for repetitive data labeling tasks. These metaprompts are core to labeling structured or unstructured text, which in turn is a crucial step in many real world scenarios and has direct value for numerous applications are areas such as business analytics, user preference modeling, content recommendation, and dialogue systems.

Workshop on Instruction Tuning and Instruction Following at NeurIPS 2023.



(a) Prompt length vs. accuracy

(b) Accuracy scores

Figure 1: Robustness of GPT-3.5 vs. GPT-4 on one of the classification tasks of Section 4. GPT-4 has much less variance in terms of accuracy (b). The input length is not well correlated with performance, making it possible to compress prompts without much loss in accuracy (a).

With larger amounts of data, however, the use of LLMs for data annotation can become quite expensive and redirects some of the importance of predictive accuracy to the cost of running the metaprompt. Moreover, new generations of LLMs such as GPT-4 are much more flexible in how prompts can be phrased or formatted as Figure 1 shows. As such, we argue that it is both important and feasible, with current generation and future language models, to optimize prompts for cost as well as predictive performance.

Although there has been prior work on prompt optimization (e.g. Zhou et al. (2023); Pryzant et al. (2023)), there are a few important challenges with metaprompts that have not to date been adequately addressed. First, many existing methods focus on low-level rewrites of the task instructions and thus are not optimizing the prompt's higher level structure. Second, almost all existing methods focus on optimizing in the context of a single example annotation instead of minibatching, which is inherently inefficient. Third, these methods focus by definition only on the predictive accuracy itself and ignore costs.

There has also been recent work on prompt compression. However, the main focus has been increasing efficiency in scenarios where the practitioner has whitebox access to the model and ability to learn/finetune model weights. This work includes prompt tuning and prompt distillation, which aim to increase efficiency of repeated tasks without having to fine tune task-specific models (eg. Lester et al. (2021); Wingate et al. (2022)). These methods learn new weights for shorter "soft prompts" that will be used instead of the longer metaprompt, but since they learn auxiliary models they cannot be applied in cases where there is only API access. Moreover, existing methods have focused on compressing model *input*, and to our knowledge, there has been no work on optimizing *output* length (even though the current cost of output tokens is roughly double that of input tokens).

To overcome these limitations, we propose a structure-aware multi-objective metaprompt optimization framework (SAMMO) that unifies prompt compression and traditional prompt optimization. SAMMO addresses shortcomings by (i) representing metaprompts as a structured objects allowing a much richer set of transformations to be searched over; and (ii) explicitly considering multiple objectives in order to balance cost and accuracy. Drawing from approaches for neural architecture search, SAMMO carries out a genetic search over a set of mutation operators that can change the structure and information contained in non-trivial ways. SAMMO is a general framework that encompasses several prompt optimization and compression techniques as special cases. It allows practioners to easily explore and automatically optimize metaprompt candidates by either relying on a pre-configured set of mutation operators or by defining task and domain-specific operators. Empirically, we show on a range of annotation tasks that SAMMO succeeds in finding metaprompts that have over 30% fewer tokens while still as accurate as the baseline prompt.

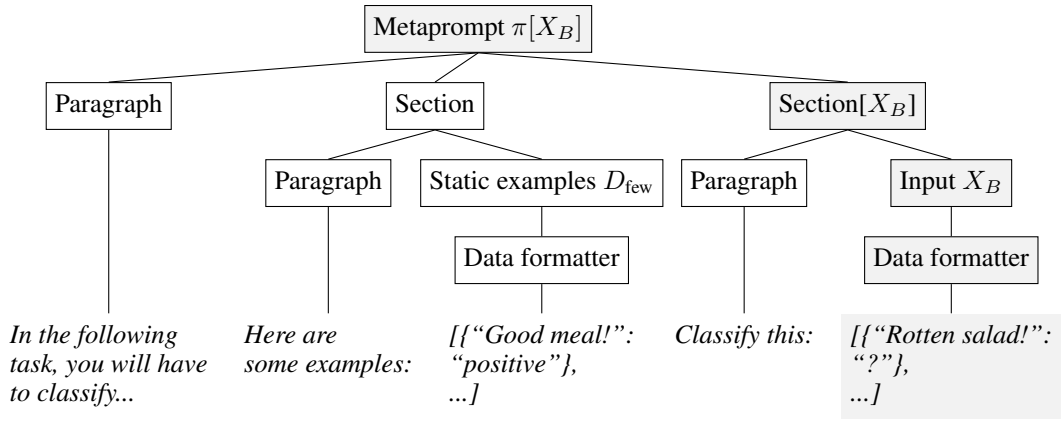


Figure 2: An instantiated metaprompt for a review classification task. The metaprompt is a tree of individual components, some of which are computed dynamically based on the minibatch B to label. Here, dynamic components are marked in light gray.

2 Problem Definition & Notation

Our goal is to automatically optimize the performance of a metaprompt π by modifying its structure, attributes, and content. Specifically, we consider K efficiency and predictive performance objectives such as accuracy and token length, expressed as a vector-valued scoring function $S(\pi, D) \in \mathbb{R}^K$, which depend on the metaprompt π and a set of labeled input data $D = (x_1, y_1), \ldots, (x_n, y_n)$.

Motivated by real-world needs, we make the following assumptions:

1. We only have small amounts of data for optimization (~ 200 examples for each task). This represents a reasonably small amount of data that can be hand-labeled by a single person; increasing the amount would limit the applicability in practice substantially.
2. The language model is a closed black box and does not output probabilities, reflecting recent changes in how access to LLMs is provided. However, we can sample from it: $\mathbf{y} \sim \text{LLM}(Y \mid \pi[X])$.
3. A scalarization function $\phi(S) \in \mathbb{R}$ is provided that specifies the trade-offs between individual quality objectives S_i. Let $S_\phi = \phi(S)$.

More precisely then, our goal is to find a metaprompt π with maximal score across the entire data distribution:

$$\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}_{D_s \sim P(X,Y)}[S_\phi(\pi, D_s)]. \tag{1}$$

Here, $P(X, Y)$ is the data distribution that the labeled samples D_s are drawn from. Since the data distribution is unknown and the evaluation cost prohibitive, we instead resort to the following sampled score in the spirit of empirical risk minimization (ERM):

$$\hat{\pi} = \arg\max_{\pi \in \Pi} S_\phi(\pi, D_{\text{train}}). \tag{2}$$

The size of the metaprompt search space is exponentially large—more precisely, it is upper-bounded by $|token\ vocabulary|^{T_{\text{max}}}$, where T_{max} is the size of the longest input a model can handle. To guide and accelerate the optimization process, we instead search over a set of higher level metaprompt structures Π representing all valid metaprompt structures.

2.1 Metaprompts

Formally, we use a dynamic function graph $\pi[\cdot] = (V, E)$ to define the structure and content of a metaprompt. The function takes a batch of input data $X_B = (x_1, \ldots, x_k)$ and maps it to a single string X'_B to be fed to the LLM, i.e., $Y_B = \text{LLM}(\pi[X_B])$. The metaprompt structure is a tree made up from individual components (e.g., sections, paragraphs), each of which call their child nodes with

Algorithm 1 Outline of SAMMO

Require: Set of mutators M, training set D_{train}, baseline prompt π_0, objective S_ϕ
$\Pi_{\mathcal{C}} \leftarrow$ INITIALIZECANDIDATES(π_0)
while *condition* **do**
  $\Pi_{\text{parents}} \leftarrow$ SELECTPARENTS($\Pi_{\mathcal{C}}, D_{\text{train},S_\phi}$)
  $\Pi_{\text{children}} \leftarrow \emptyset$
  for each $\pi_{\text{par}} \in \Pi_{\text{parents}}$ **do**
    $M_{\text{active}} = \{m$ can be applied to $\pi_{\text{par}} \mid m \in M\}$
    $M_{\pi_{\text{par}}}$ = CHOOSEMUTATORS($M_{\text{active}}, \Pi_{\text{parents}}$)
    Add $\bigcup_{m \in M_P}$ MUTATE(m, Π_{parents}) to Π_{children}
  end for
  $\Pi_{\mathcal{C}} \leftarrow \Pi_{\mathcal{C}} \cup$ SELECTCHILDREN($\Pi_{\text{children}}, D_{\text{train}}, S_\phi$)
end while
return best candidate from $\Pi_{\mathcal{C}}$

the input B. Each node is a function $v = f_{t,a}[\cdot]$ with text t and other attributes a. We call v *static* if it does not depend on the input data B; otherwise, it is a *dynamic* node with notation $v[\cdot]$. We will sometimes refer to subtrees containing only text as *task instructions*.

To give some intuition for what this looks like in practice, Figure 2 shows a metaprompt for a review classification task which has already been instantiated with some input data B. It has a section with task instructions at the beginning, followed by a set of static in-context examples (although they could be selected dynamically as well) and the unlabeled input data. Note that the data format is explicitly represented—here it serializes data to a JSON array with each example being a dictionary. We also note the similarity with neural architectures—static components can be seen as global parameters, dynamic components correspond to projection operators or activation layers as they operate on the input X_B. With this notation, we can see how searching over Π in Eq. 2 amounts to searching over different graph structures (V, E) and node attributes t, a.

3 SAMMO: Structure-aware Multi-Objective Metaprompt Optimization

We outline SAMMO, which is a general framework for optimizing the performance of metaprompts. SAMMO uses evolutionary search with a rich set of mutating operators to explore the space of prompts, extending previous approaches such as Automatic Prompt Engineering (APE) Zhang et al. (2022b) in two important dimensions. First, we represent metaprompts as structured objects which in turn allows us to do novel transformations such as changing the output format. Second, we consider multiple objectives when optimizing prompts to balance important properties such as input and output length with predictive accuracy.

3.1 SAMMO Search Procedure

Algorithm 1 shows an outline of how SAMMO works on a high level. It takes a initial baseline metaprompt (π_0) and small amount of training data as input, and uses evolutionary search for combinatorial optimization over the multiple objectives specified in S_ϕ. The search is initialized with π_0 and SAMMO uses a set of specified mutators to iteratively modify the current metaprompt candidates and select those with the best objective on the training data.

Algorithm 1 can be used with a variety of search algorithms. Specific choices of the INITIALIZECANDIDATES, SELECTPARENTS, *condition*, and SELECTCHILDREN operations yield common search algorithms such as beam search, regularized evolutionary search Real et al. (2019) or breadth-first search. For example, beam search typically starts with one candidate and then selects all parents at the current depth, keeping only the top k performing children for the next round. The novelty of SAMMO lies not in this outer search but in the fact that (i) we represent the higher level structure of a metaprompt explicitly which in turn allows us to a rich set of operators that can both transform the structure as well the content and (ii) now can learn on a operator level which ones should be prioritized in the search process. Below we give more details on each.

Table 1: Examples for mutation operators, grouped by what part of a metaprompt π they affect. SAMMO allows for a rich set of operations whereas traditional prompt optimization techniques have only focused on operations that change the text.

Type	**Operator**	Description
Text t	PARAPHRASE	Rewrite to keep meaning
	INDUCEINSTRUCTIONS	Generate instructions from examples
	SHORTENTEXT	Reduce length to certain number of words
	TEXTTOBULLETPOINTS	Turn into bullet list
	REMOVESTOPWORDS	Filter out stopwords
Attribute a	CHANGESECTIONFORMAT	How sections are rendered (e.g., markdown, XML)
	CHANGEDATAFORMAT	How data is rendered (e.g., JSON, XML)
	DECREASEINCONTEXTEXAMPLES	Resample a smaller number of examples
Structure (V, E)	DROPSECTION	Remove a section
	REPEATSECTION	Repeat a section somewhere

Algorithm 2 Prior-Informed Mutator Sampling

Require: Set of mutators M with counts of $(n_{\text{improved}}, n_{\text{tried}})$ with prior $p = (p_\alpha, p_\beta)$,
1: $\alpha = n_{\text{improved}} + p_\alpha$
2: $\beta = n_{\text{tried}} - n_{\text{improved}} + p_\beta$
3: $w_m = \arg\max \text{Beta}(\alpha, \beta) \;\; \forall m \in M$
4: normalize w s.t. $\sum_m w_m = 1$
5: **return** sample(s) from Multinomial(w)

3.2 Mutation Operators

Formally, a mutation operator is a function $m : \Pi \longrightarrow \Pi$ that maps a metaprompt π to an edited version π'. This *structure-aware* component of SAMMO opens up a new and exciting class of operators. These can range from trivial (e.g., rephrasing a sentence) to complex (e.g., inducing a new set of task instructions). Table 1 shows a non-comprehensive set of mutation operators, grouped by what part of a metaprompt they change. To the best of our knowledge, SAMMO is the first optimization method that can also optimize for large structural changes and data formatting.

3.3 Operator Selection

During the search process, SAMMO needs to select a subset of mutation operators from the set of all applicable mutation operators for each candidate. One might be tempted to frame this as a multiarmed bandit problem, however, best-arm (mutation operator) identification would lead to the algorithm converging on using only a single mutation operator which is clearly undesirable. Instead, we use a Beta-Bernoulli distribution to model successes in improving the objective of each mutation operator and then draw from a Multinomial distribution with probabilities proportional to the maximum a-posteriori success rate estimates from each mutation operator. The exact steps are given in 2. We call the resulting method $\text{SAMMO}_{\text{prior}}$ and also compare it with $\text{SAMMO}_{\text{uni}}$, a variant which uses uniform sampling in the experiments below.

3.4 Specializations of SAMMO

The following methods are examples for special cases of SAMMO:

- APE (Zhou et al., 2023): Having INITIALIZECANDIDATES sample via the templates provided by APE with a single mutation operation that paraphrases a candidate yields the well-known APE.

- STDC (Yin et al., 2023a): Starts with a single baseline metaprompt π_0 and uses sequential search, maintaining only a candidate set Π_C of size one. The mutation operator looks at the current level of the syntax tree of the instructions and prunes a node if it does not decrease accuracy.

4 Experiments

In our experiments, we optimize four objectives:

1. maximize the accuracy on the classification task,
2. minimize the number of tokens in the input,
3. minimize the number of tokens in the output,
4. minimize the rate of parse errors.

We also use a linear scalarization of the overall objective,

$$S_\phi(\pi, D) = \sum_{i=1}^{K} w_i S'_i(\pi, D) \tag{3}$$

Unless otherwise noted, we use uniform weights ($w_i = 1$). Individual objectives will use the performance of a baseline metaprompt π_0 on the training data as a reference:

$$S'_i(\pi, D) = \frac{S_i(\pi, D) - S_i(\pi_0, D_{\text{train}})}{S_i(\pi_0, D_{\text{train}})}. \tag{4}$$

Intuitively, this yields the relative percentage of improvement with respect to the baseline metaprompt. While there are many other possible scalarization schemes, we settled on linear scalarization due to its conceptual simplicity and flexibility in trading off objectives. Scalarization schemes based on L_p metrics for large p will heavily penalize solutions that fail to improve just a single dimension (Miettinen, 1999).

4.1 Methods & Baselines

Selecting a set of appropriate baselines for metaprompt optimization is challenging for two reasons. First, most existing work does not handle minibatching; second, we work under the assumption that the language model is closed and can only be sampled from – in particular, we cannot employ fine-tuning or rely on token probabilities. We were able to find two previously published methods that were applicable with only minor modifications, APO and STDC. Below are the details of all methods:

- **APO: Automatic Prompt Engineering (Zhou et al., 2023)**: The core idea here is to feed the model labeled examples and let it come up with a set of instructions. We induce a set of initial instructions via the templates that the authors provide and perform beam search with the PARAPHRASE mutation operator as in their paper.
- **STDC: Syntax-guided Task Definition Compression (Yin et al., 2023a)**: This approach builds a syntax parse tree from the task instructions and removes sub-trees in bread-first order if they do not decrease the accuracy of the prompt.
- **SAMMO**: We use beam search as the backbone of Algorithm 1 and initialize Π_C to only the baseline prompt $\{\pi_0\}$. We use all mutation operators listed in Table 1 as possible operations, and choose mutators either uniformly at random (SAMMO$_{\text{uni}}$), or via the prior-informed sampling of Algorithm 2. We use the success counts from the previous runs of SAMMO$_{\text{uni}}$.

For each method, we set a budget of 48 candidate evaluations. For STDC, this means 48 sequential steps; for APO we split it into 24 initial candidates and 24 subsequent ones. For SAMMO, we use beam search with width four and depth six, and let each mutator generate two mutations.

Table 2: Averages across 10 tasks using GPT-3.5 (16k) as a backend model, best values in bold. SAMMO compresses both the number of input and output tokens while maintaining similar accuracy levels as the baseline prompt π_0.

	train					test				
method	obj	acc	input	output	parse	obj	acc	input	output	parse
Baseline π_0	0.00	0.56	12992	2267	0.98	0.01	**0.58**	12241	2377	**0.97**
APE	0.11	0.52	9516	589	0.97	0.10	0.53	9288	702	**0.97**
STDC	-0.13	0.59	10637	2108	**1.00**	-0.27	0.54	9806	3046	0.96
SAMMO$_{\text{uni}}$	**0.19**	**0.59**	**9039**	**486**	0.96	**0.17**	0.56	**8488**	**533**	0.95
SAMMO$_{\text{prior}}$	0.18	0.57	9764	591	0.92	0.14	0.55	9364	657	0.90

Table 3: Averages across 10 tasks using GPT-4 (32k) as a backend model. SAMMO again achieves the highest compression rates under comparable accuracy.

	train					test				
method	obj	acc	input	output	parse	obj	acc	input	output	parse
Baseline π_0	0.00	0.76	12992	600	**1.00**	0.00	**0.75**	12747	601	**1.00**
APE	0.08	0.76	9972	603	**1.00**	0.06	0.72	9727	602	**1.00**
STDC	0.05	**0.78**	10971	600	**1.00**	0.04	0.74	10726	600	**1.00**
SAMMO$_{\text{uni}}$	0.16	0.77	8708	455	**1.00**	0.15	0.72	8364	450	0.99
SAMMO$_{\text{prior}}$	**0.22**	0.77	**8275**	**321**	0.98	**0.20**	0.73	**7726**	**315**	0.96

4.2 Tasks

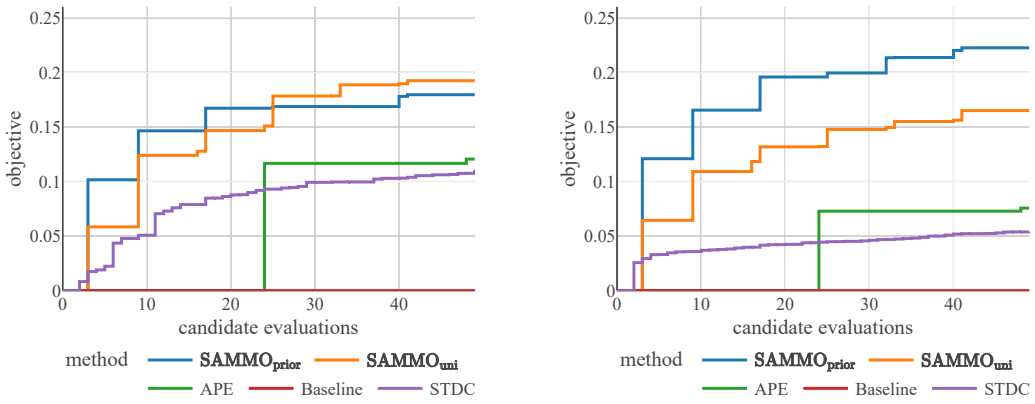
In this paper, our focus is on tasks that require extensive guidelines for annotation. To find appropriate tasks, we sampled ten tasks from the Super-NaturalInstructions benchmark (Wang et al., 2022) which (i) were in English, (ii) had more than 200 labeled examples, (iii) had less than ten possible output labels so they can be treated as classification problems and (iv) had instructions of 1000 characters or above. We sample training and test sets of sizes 100 each, plus a set of five in-context examples. Our baseline prompt π_0 uses the instructions provided with a task followed by five in-context examples. We use the newline delimited format of Cheng et al. (2023) to render batched data.

4.3 Results: Overall Performance

Tables 2 and 3 show the the average performance across the ten tasks with either GPT-3.5 or GPT-4 as a backbone LLM. We report both the overall objective (obj) as well as the four individual objectives: accuracy (acc), input length in tokens (input), output length and parse success rate.

Starting with the results on GPT-3.5 in Table 2, we can see that both SAMMO variants outperform all other baselines substantially in terms of the overall objective. They also manage to compress both input and output by a considerable amount – SAMMO$_{\text{uni}}$ reduces input length by 30% and output length by almost 78% while seeing very little drop in accuracy. Interestingly, STDC actually results in an increased output length; this is due to the model adding explanations or repeating parts of the input in the face of uncertainty. Between the two variants, SAMMO$_{\text{prior}}$ with prior-informed sampling does slightly worse than uniform sampling; we return to this difference in the next subsection.

On GPT-4, most trends are similar, albeit with higher average accuracy across all methods. Again, we find that SAMMO achieves respectable compression rates, all while maintaining high accuracy. SAMMO$_{\text{prior}}$ manages to compress the number of output tokens by another 50% compared to GPT-3.5, mostly due due to greater overall LLM flexibility and robustness. Moreover, it also manages to compress the input even more – over 36% compared to the baseline metaprompt. However, one key difference is that SAMMO$_{\text{prior}}$ outperforms uniform sampling on this benchmark.



(a) GPT-3.5

(b) GPT 4

Figure 3: Learning curves of all methods showing the training objective vs. the number of evaluated candidates. SAMMO increases performance quickly and continues to improve throughout.

4.4 Results: Learning curves

Figure 3 shows the learning curves of all methods in terms of the training objective over the number of candidate evaluations. Note that many methods employ parallel stages, and so we plot the best current value of S_ϕ after the parallel initialization or mutation stage has been completed (cf. lines 1 and 10 in Algorithm 1). SAMMO manages to optimizes the objective quickly and continues to see improvements as the search deepens. APE sees the biggest jump in performance after the candidate generation stage and sees only small gains during paraphrasing which is in line with the authors' observations.

5 Related Work

Related work can be categorized into two areas: prompt optimization and prompt compression.

In prompt optimization, the main focus in on improving the accuracy of the prompts. Past work has typically focused on simpler (eg. single task) prompts with less structure. Automatic Prompt Engineer (APE) Zhou et al. (2023) uses beam search, generating instruction candidates from just input-output pairs, followed by rewriting steps. Automatic Prompt Optimization (APO) Pryzant et al. (2023) re-writes instructions by generating explanations for errors, changing the prompt to reflect explanations, and then generating more prompt candidates by paraphrasing. GrIPS Prasad et al. (2022) uses beam search with edit, add and delete operations on the syntax tree after parsing the instructions. InstructZero Chen et al. (2023) optimizes the prompt locally via Bayesian Optimization using a smaller surrogate model and uses calls to LLM API as feedback.

In prompt compression, the main focus has been on *prompt tuning* to increase efficiency of repeated tasks without having to fine tune task-specific models (eg. Lester et al. (2021)). This allows to use of frozen LLMs, but requires learning a separate sets of weights for the "soft prompt" tokens that will be appended for each task (eg. Chevalier et al. (2023)). Following this direction, there are other works that focus on prompt distillation or meta finetuning. For example, Wingate et al. (2022) uses a distillation objective to steer text generation with compressed instructions and Mu et al. (2023) use meta-learning to compresses prompts into "gist" tokens.

When practitioners only have access to LLMs through an API, they do not have the ability to access the probability distribution of the output tokens or the gradient information. In this case, a method to compress the discrete choice of tokens in a prompt is needed. Jung & Kim (2023) use a reinforcement learning approach to learn which tokens can be excluded from a prompt without degradation in accuracy. However, this requires model-specific tokenization knowledge that may also be opaque to API users.

Yin et al. (2023b) specifically considers prompts with complex task definitions with different types of content. They propose a syntax-guided compression algorithm that learns which instruction components can be excluded without drop in accuracy.

6 Conclusion

In this paper, we introduce a new framework, Structure-aware Multi-Objective Metaprompt Optimization (SAMMO), to make annotating large amounts of data via LLMs more viable. This framework is designed to balance multiple objectives efficiently, taking both operational costs as well as task performance into consideration.

SAMMO represents metaprompts as dynamic function graphs, and employs a set of mutation operators to alter the structure and content of metaprompts. This approach notably outperforms existing methods of prompt optimization, as demonstrated through several annotation tasks in our experimental evaluation.

The experimental results consistently show that SAMMO can substantially improve the efficiency and predictive accuracy of metaprompts on LLM-based annotation tasks, with a 30% or more reduction in token usage. Our results further show that SAMMO works efficiently with small search budgets and performs robustly regardless of whether the underlying LLM was GPT-3.5 or the more powerful GPT-4.

Future research will further explore the potential of this structure-aware method to optimize complex metaprompts. Additional mutation operators could be considered as well as more complex annotation output structures. We also anticipate future adaptations of our method to yield not only a single solution, but a set of candidates that are on the Pareto front.

References

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

Zhoujun Cheng, Jungo Kasai, and Tao Yu. Batch prompting: Efficient inference with large language model apis. *arXiv preprint arXiv:2301.08721*, 2023.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. *arXiv preprint arXiv:2305.14788*, 2023.

Hoyoun Jung and Kyung-Joong Kim. Discrete prompt compression with reinforcement learning. *arXiv preprint arXiv:2308.08758*, 2023.

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

Kaisa Miettinen. *Nonlinear multiobjective optimization*, volume 12. Springer Science & Business Media, 1999.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens. *arXiv preprint arXiv:2304.08467*, 2023.

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

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

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Tianxiang Sun, Zhengfu He, Hong Qian, Yunhua Zhou, Xuanjing Huang, and Xipeng Qiu. Bbtv2: Towards a gradient-free future with large language models, 2022a.

Tianxiang Sun, Yunfan Shao, Hong Qian, Xuanjing Huang, and Xipeng Qiu. Black-box tuning for language-model-as-a-service. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 20841–20855. PMLR, 17–23 Jul 2022b. URL https://proceedings.mlr.press/v162/sun22e.html.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, et al. Super-naturalinstructions:generalization via declarative instructions on 1600+ tasks. In *EMNLP*, 2022.

David Wingate, Mohammad Shoeybi, and Taylor Sorensen. Prompt compression and contrastive conditioning for controllability and toxicity reduction in language models. *arXiv preprint arXiv:2210.03162*, 2022.

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

Fan Yin, Jesse Vig, Philippe Laban, Shafiq Joty, Caiming Xiong, and Chien-Sheng Wu. Did you read the instructions? rethinking the effectiveness of task definitions in instruction learning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*. Association for Computational Linguistics, 2023a.

Fan Yin, Jesse Vig, Philippe Laban, Shafiq Joty, Caiming Xiong, and Chien-Sheng Jason Wu. Did you read the instructions? rethinking the effectiveness of task definitions in instruction learning, 2023b.

Tianjun Zhang, Xuezhi Wang, Denny Zhou, Dale Schuurmans, and Joseph E. Gonzalez. Tempera: Test-time prompting via reinforcement learning. 2022a.

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

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

A Appendix

The table below organizes related work across serveral dimensions.

Table 4: Related Work

	space	candidate generation	optimization method	uses logits	tasks	notes
APO (Pryzant et al., 2023)	discrete	error explanations, instruction correction, paraphrasing	beam search + bandit	no	class.	
GRIPS (Prasad et al., 2022)	discrete	ops on instruction subphrases	beam search	no	class.	
Tempera (Zhang et al., 2022a)	discrete	ops on instruction subphrases	RL	yes	general	
APE (Zhou et al., 2023)	discrete	inverse instruction generation, paraphrasing	beam search	yes	general	requires in-place generation
Xu et al. (2022)	discrete	backtranslation, paraphrasing	evolutionary algorithm	yes	general	
InstructZero (Chen et al., 2023)	cont.	subspace sampling	Bayesian	no	general	
Sun et al. (2022b)	cont.	subspace sampling	evolutionary algorithm	yes	class.	
Sun et al. (2022a)	cont.	subspace sampling	layer-wise evolutionary	yes	class.	layer-wise logits needed