creative commons

COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

Ph.D. DISSERTATION

Leveraging Contexts for Efficient Automatic Prompt Engineering on Large-scale Language Models

대규모 언어 모델에서 효율적인 자동화된 프롬프트 엔지니어링을 위한 맥락의 활용

AUGUST 2023

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Hyeonmin Ha

Leveraging Contexts for Efficient Automatic Prompt Engineering on Large-scale Language Models

대규모 언어 모델에서 효율적인 자동화된 프롬프트 엔지니어링을 위한 맥락의 활용

지도교수 전 병 곤

이 논문을 공학박사 학위논문으로 제출함

2023년 8월

서울대학교 대학원

컴퓨터 공학부

하 현 민

하현민의 공학박사 학위논문을 인준함

2023년 6월

위 원 장	황승원	(인)
부위원장	전병곤	(인)
위 원	윤성로	(인)
위 원	김건희	(인)
위 원	서민준	(인)

Abstract

Prompting has gained tremendous attention as an efficient method for the adaptation of large-scale language models (LMs). A prompt typically has a task description and demonstration examples and is fed as input to an LM. Then, the LM learns the task from the context given by the input and processes queries. This phenomenon is called In-context Learning. However, prompts often act against human intuition and report unstable performances, which has motivated methods that automatically find effective prompts.

Automatic prompt tuning methods have shown promising performances on various NLP tasks but still fall behind several LM adaptation methods, such as full-parameter fine-tuning, in some scenarios. Despite the sub-optimal performances, unique capabilities of prompting, such as simplicity or parameter efficiency, encourage improving prompting methods.

Moreover, even though the prompt tuning methods are effective in LM adaptation, they are not designed to support language-model-as-a-services (LMaaSs). Recent large-scale LMs typically provide their capabilities via services. Such Language-Model-as-a-Services (LMaaSs) have unique constraints for practical deployments compared to in-house models; the model's internal parameters are not publicly open. This requires users to prepare task-specific prompts for in-context learning when using the service. However, LMaaSs does not provide automatic prompt tuning methods because of their heavy computation overheads. An LMaaS user may use several black-box prompting methods that do not require parameter access or service providers' additional support, but it is really hard for non-expert users to deploy and execute such methods on their

in-house machines.

In this thesis, we first propose a novel regularization method, CoRe, for gradient-based prompt tuning techniques, which guides a prompt to produce a task context properly. CoRe realizes two regularization effects — context attuning and context filtering — that improve prediction performance in a zero-shot in-context learning setting where a model makes inferences only with the prompt tuned by CoRe, without any demonstration examples for in-context learning. Context attuning guides the context generated by the input and the tuned prompt toward embedding the appropriate context for the task. In our theoretical analysis, regularizing the context extends to improving zero-shot in-context learning performance. Context filtering steers the prompt to select only the task-related context so that context attuning solely focuses on creating and sending the right task context. We evaluate CoRe on natural language understanding datasets and two large language models, GPT2-XL and GPT-J. Our training scheme shows performance improvements up to 11.9% on GPT2-XL, and up to 6.3% on GPT-J in zero-shot settings.

We then propose MetaL-Prompt, a novel lightweight prompt generation method for LMaaS based on meta-learning. MetaL-Prompt makes a prompt generation model (PGM) which generates a task-specific prompt from few-shot examples of an arbitrary user task without additional training during service. We also suggest trainable padding to mitigate the overhead from the generation process of the meta-learning, and explore generation of diverse prompt types using the prompt generation model. MetaL-Prompt is compute-efficient since the PGM extracts task information from the context caused by the concatenation of the few-shot examples, and generates a corresponding prompt in a single forward pass. Therefore, MetaL-Prompt introduces negligible computation overheads when deployed on LMaaSs, and the services can support

a tremendous number of various tasks with automatically generated prompts with MetaL-Prompt. We evaluate MetaL-Prompt in diverse meta-learning settings, and it improves the performance up to 19.4% for averaged F1 score on unseen QA datasets in a zero-shot in-context learning setting compared to the state-of-the-art baseline, even with much lower computation costs.

Keywords: prompt tuning, prompt, in-context learning, language model

Student Number: 2016-21234

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

In recent years, many large-scale deep learning language models (LMs) have been released, and have demonstrated outstanding performances on diverse natural language processing (NLP) tasks. Such models learn language modeling on plain corpora (e.g., articles, news, or books), but after the learning, the models can be transferred to specific NLP tasks using adaptation methods such as fine-tuning.

One interesting method for LM adaptation is in-context learning [54, 4]. In-context learning adapts an LM to a task by giving a special text, usually called a prompt, including an explanation or some demonstration examples for the task. The LM solves the problem following the context given by the prompt. In-context learning (ICL) has unique capabilities in that it does not mandatorily require training examples or parameter access for LM adaptation.

Initially, a prompt is manually crafted by a developer with human intuition.

However, because of the unstable performances and unpredictable behaviors of manual prompts [42], automatic prompt tuning methods have been proposed. The automatic methods have shown remarkable performances compared to hand-crafted prompts cite, but still fall behind full-parameter fine-tuning [16]. This motivates further improvements in automatic prompt tuning.

Automatic prompt tuning methods also have another challenge in practicality. Recently, many companies provide large language models' capabilities as services. Such Language-Model-as-a-Services (LMaaSs) support diverse user tasks with in-context learning from prompts (i.e., instructions and demonstrations of the task). However, for the users, manual crafting of prompts or running automatic prompt tuning methods by themselves is demanding. Despite such challenges, LMaaSs do not provide automatic prompt tuning methods on the services. One of the major obstacles to deploying them on an LMaaS is their heavy computation costs. Such prompt tuning methods are typically designed to iterate tens of thousands of examples For example, P-tuning [42], which is a representative gradient-based prompt tuning method, executes forward and backward passes of about 50K sequences to optimize a prompt on SuperGLUE RTE [63]. BBTv2 [61] optimizes a prompt with forward passes on 250K sequences for instance. Such heavy computations are unaffordable overheads for LMaaSs.

1.2 Contributions

In this thesis, we propose two prompt tuning methods for improving the performances of gradient-based prompt tuning methods, and lightweight prompt tuning. Both methods leverage contexts that appear when concatenated examples are fed to an LM.

We first propose a novel gradient-based prompt tuning method, CoRe, which guides a prompt to properly produce a task context. CoRe realizes two regularization effects — context attuning and context filtering. Those improve prediction performance in a zero-shot ICL where additional demonstration examples are not provided, and only prompt and test examples are available for prediction with ICL. Context attuning guides the context generated by the input and the tuned prompt toward embedding the appropriate context for the task. In our theoretical analysis, regularizing the context extends to improving zero-shot ICL performance. Context filtering steers the prompt to select only the task-related context so that context attuning solely focuses on creating and sending the right task context. CoRe shows performance improvements up to 11.9% on GPT2-XL, and up to 6.3% on GPT-J in zero-shot ICL setting on SuperGLUE [63].

In this paper, we propose MetaL-Prompt, a novel lightweight automatic prompt generation method for LMaaSs. MetaL-Prompt meta-trains a prompt generation model (PGM) in order that an LM robustly learns from contexts (i.e., in-context learning) induced by the created prompt. Due to our meta-learning, a PGM can generate a prompt for an unseen task without additional training of the PGM for the task. Moreover, the PGM can generate the prompt from the context created by few-shot examples with a single forward pass, which requires much less computation cost compared to the previous methods. We evaluate MetaL-Prompt on diverse unseen tasks, and it improves the performance up to 19.4% for mean F1 score on QA datasets compared to the state-of-the-art baseline P-tuning, even with such a small computation cost.

1.3 Dissertation Structure

The rest of this thesis is organized as follows. Chapter 2 describes in-context learning and related prompt tuning methods. In Chapter 3, we present a gradient-based prompt tuning method, CoRe, which improves zero-shot ICL performances of the previous methods leveraging context regularization. In Chapter 4, we propose our lightweight prompt generation method, MetaL-Prompt, which generates a prompt in a single forward pass by extracting a task context from few-shot examples We conclude in Chapter 5 and suggest future works.

Chapter 2

Background

In this chapter, we first explain in-context learning, which is an adaptation method for large-scale language models (LMs). Then, we introduce automatic methods to improve in-context learning. Finally, we present how large-scale language models are deployed and utilized in the real world.

2.1 In-context Learning

Prior works have proposed that large-scale LMs are able to perform well in diverse tasks by learning from prompts that include instructions or a few demonstration examples [54, 4], even without additional tuning of the parameters. This phenomenon is called In-context Learning, and it enables LMs to process various tasks without additional training. Prompts have been used to adapt pretrained LMs to numerous downstream tasks such as Natural Language Inference [59], text classification [18, 53, 59], question answering [32, 34, 55] and many more.

A prompt is a form of task specification given as part of the input, and aims to guide the language model into generating the desired output according to the given task. A prompt usually consists of a task description and a template of the input with placeholders for data. For example, when given an input "**Translate English to Spanish:** I love you →", the LM is expected to finish the input by generating the answer "te amo". For improved performance, some demonstration examples are often provided as a training set in deep learning.

Prompting is more memory-efficient when we want the LM to process various different tasks compared to fine-tuning which is a representative LM adaptation technique. In prompting, the LM parameters are typically frozen, and developers only have to keep at most dozens of tokens per task, which are task descriptions and input templates, for LM adaptation. Even including demonstration examples, hundreds of tokens per task are kept. In fine-tuning, the parameters of the LM are updated to optimize the task performance. Developers have to keep different versions of parameters per task, which can be as huge as 540 billion parameters [8] in the status quo.

Prompting is also more simple and easier to realize in serving various tasks with a single LM than other LM adaptation methods such as LoRA [29] or adapter [28]. Such methods require layers with different parameters for each task, and this makes really hard to serve queries from diverse tasks in a single forward pass (i.e., batching the queries). However, prompting simply prepends some texts to queries, and does not require special implementation to support such scenarios.

2.2 Manual prompt tuning

In the early stages of prompting, prompts were often handcrafted with heuristics. Radford et al. [54] and Brown et al. [4] demonstrated the transferability of pretrained LMs using manual prompts. These two works show that LMs can perform well on diverse NLP tasks when provided with some adequate prompts.

In subsequent works, manual prompts have been applied to a number of research areas such as factual probing [50, 31], knowledge mining [11], question answering [34, 55], and text classification [53, 59]. Despite their remarkable adaptation ability, manually crafted prompts have exhibited unstable performances [42]. Besides, prompts that show good performance are, in fact, quite against commonsense, unable to understand why some specific prompts work well [42].

2.3 Automatic prompt engineering

In addition to the unpredictable performance and unpredictable behavior of manual prompts [42], the process of creating manual prompts requires significant human effort, as it involves devising and evaluating numerous handcrafted templates with different patterns. Consequently, recent studies have proposed techniques to automate the search for optimal prompts. As illustrated in Table 2.1, prompt engineering methods can be classified into two categories based on their usage of gradients. We can additionally categorize them according to the computation costs that they require. In Table 2.1, we assign the label "low-cost method" to approaches that are specifically designed for low computation costs (i.e., involving fewer than 500 iterations) or have demonstrated their efficacy under such constraints.

	Gradient-based	Gradient-free
High cost	P-tuning [42]	RLPrompt [13]
	SoftPrompt [36]	TEMPERA [69]
	Prefix-tuning [39]	APE [72]
	AutoPrompt [60]	BBTv2 [61]
	P-tuning v2 [43]	Clip-Tuning [5]
	CoRe (§3)	BDPL [15]
		APE [72]
Low cost	MetaPrompting [27]	Instruction Induction [24]
	MetaL-Prompt (§4)	

Table 2.1 Classification of automatic prompt engineering methods based on gradient usage and computation cost for prompt creation. Methods that are specifically designed to operate under low computation costs, with performance validated iterating fewer than 500 examples, are classified as low-cost methods.

2.3.1 Gradient-based Prompt Engineering

One popular approach in prompt engineering involves gradient-based methods, which continually update prompt tokens or parameterized prompt embeddings by leveraging gradients. Lester et al. [35], OPTIPROMPT [71], AutoPrompt [60] and P-tuning [42] initially concatenate (randomly) initialized embeddings of tokens for prompts with the embeddings of input data in a predefined order. Other methods incorporate prompts within hidden states rather than inputs. Specifically, Prefix-tuning [39] appends prompts to the keys and values of transformer layers. Similarly, P-tuning v2 [43] and BBT [62, 61] also adopt this approach. The prompts are directly optimized using SGD on a given task dataset. The optimized prompts are subsequently utilized during inference

without any modification.

Certain methods have focused on improving prompt performance by enhancing the initialization process. P-tuning v2 [43] employed multi-task learning to achieve this objective. MetaPrompting [27] utilized meta-learning techniques, specifically MAML [17] and its variants [48, 1], which are methods designed to enhance the initialization of deep learning models for few-shot learning scenarios. MetaPrompting also confirmed its effectiveness with several steps of training which is a low-cost setting.

While gradient-based prompting methods have shown promising performances across various NLP tasks, they have demonstrated degraded performance in some cases compared to other adaptation methods such as fine-tuning or LoRA [29], as discussed in Ding et al. [16]. These findings serve as a motivation for further advancements and improvements in prompting methods.

2.3.2 Gradient-free Prompt Engineering

Gradient-based prompt engineering methods have demonstrated impressive performance in LM adaptation. However, they often require substantial memory space for backward passes. Additionally, certain language models are offered as black-box services (as discussed in Section 2.4), where access to model parameters is disabled. Consequently, gradient-based methods are not applicable to such models. These limitations have led to the development of gradient-free prompt tuning methods as an alternative approach.

Prompt generation with pretrained language models

One simple method is generating a prompt using an LM as illustrated in Figure 2.1 (1). Gao et al. [18] utilize the pretrained T5 model [55] to complete the missing spans of an input sentence and construct a prompt. PromptBoost-

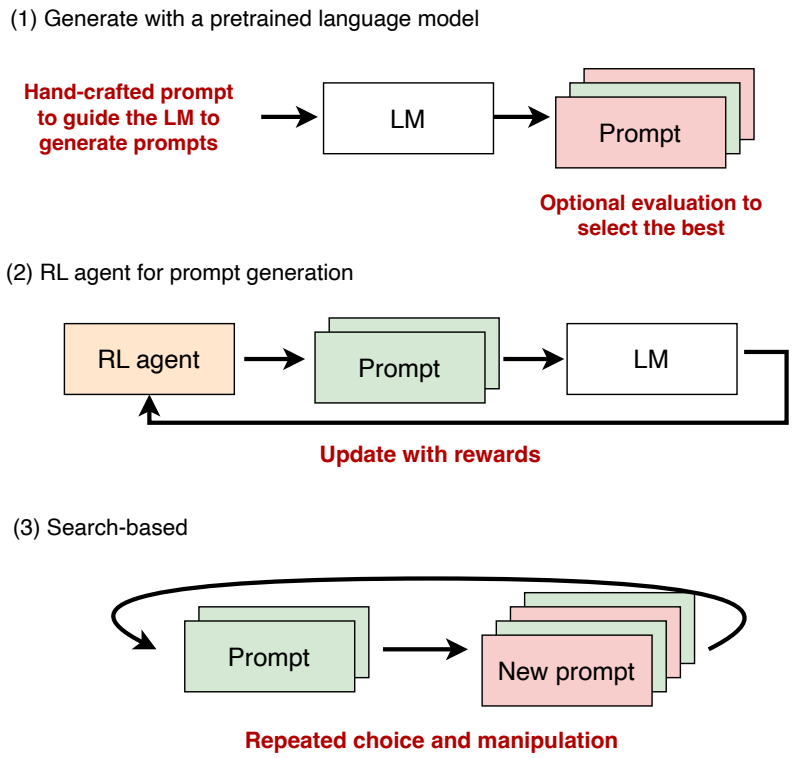

Figure 2.1 Illustration of three categories of gradient-free methods.

ing [26] further enhances performance by ensembling prompts generated by Gao et al. [18]. Instruction Induction [24] employs an LM to generate a natural language instruction for the target task, utilizing few-shot demonstrations and a hand-crafted instruction that guides the LM in generating the task instruction. While Instruction Induction generates a prompt efficiently through a single generation process, the performance of the generated prompt is not satisfactory. Consequently, Automatic Prompt Engineer [72] improves upon this method by iteratively generating candidate instructions using an LM and evaluating the candidates to identify the best prompt.

Prompt generation with RL agents

RLPrompt [13] is another approach that involves generating prompts using an LM, but it employs reinforcement learning to guide the LM in prompt generation as depicted in Figure 2.1 (2). RLPrompt adapts a pretrained LM using reinforcement learning (RL) to generate prompts tailored to specific tasks. In the training process, RLPrompt receives rewards from a black-boxed target LM without directly accessing its parameters. It is important to note that this method requires an additional pretrained LM that can access the parameters and is specifically adapted for prompt generation. TEMPERA [69] also utilizes RL but takes a different problem. It trains an RL agent to modify an existing prompt for a given task. During testing, the RL agent, trained by TEMPERA, receives an initial prompt containing instructions and demonstration examples. The agent then applies pre-defined editing actions, such as swapping or deleting phrases, to refine the prompt according to the specific requirements of the task.

Search-based approaches

Some methods adopt more sophisticated optimization methods: search-based optimization (Figure 2.1 (3)). GRIPS [52] repeatedly edits prompts starting from a hand-crafted prompt and searches the best prompt by evaluating the edited prompts. BBT [62] and BBTv2 [61] adopt evolutionary search on a continuous prompt (or soft prompt), which is a continuous vector like word embeddings, instead of a natural language prompt which is mapped to discrete tokens. BBTv2 improves this by adopting the prompt design of Prefix-tuning [39], which prepends prompts to keys and values of transformer layers. Clip-Tuning [5] also takes an evolutionary search to optimize a continuous prompt as BBT, but Clip-Tuning uses fitness scores from diverse subnetworks of an LM to diversify

the data representations.

Estimated gradients

BDPL [15] optimizes prompts as gradient-based prompt tuning, but uses estimated gradients obtained from a variance-reduced policy gradient algorithm.

2.4 Language Model as a Service

According to the emergence of large-scale language models (LMs), several companies train such models, and provide them as services [49, 19, 10], which is often called Language-Model-as-a-Service (LMaaS) [62]. On an LMaaS, a user sends an input text to the service via the API, then the user can obtain some corresponding prediction results from the LM. A model of an LMaaS is usually black-boxed due to safety to avoid potential malicious misuse, commercial reasons, etc, which disables users to tune the model via conventional training methods (i.e, fine-tuning) for their tasks if the service provider especially supports such method in the black-box. Even if such tuning methods are absent, the services can support various and unique tasks of users using in-context learning, which we mentioned in Section 2.1, where an LM learns the user task from a carefully crafted input.

2.5 Computation costs of automatic prompt engineering methods

As we mentioned in the previous section, capabilities of large-scale language models (LMs) are often provided via black-boxed services, LMaaS. One of the most popular method to adapt an LM on an LMaaS to a user's task is in-context learning, which gives a prompt that includes a (natural language) task instruc-

Method	# of examples
P-tuning [42]	56,000
SoftPrompt [36]	960,000
RLPrompt [13]	96,000 – 192,000
TEMPERA [69]	65,536
MetaPrompting [27]	240
APE [72]	12,800
BBTv2 [61]	256,000
Clip-Tuning [5]	16,000 – 32,000
BDPL [15]	32,000 – 128,000
MetaL-Prompt (ours)	16

Table 2.2 Number of examples that each prompt tuning methods processed for the experiments in the original papers. Note that we present the additional costs to support one more dataset to show scalabilities of the baselines. MetaPrompting and MetaL-Prompt (ours) requires meta-learning but it is conducted once not for each task.

tion or/and demonstration examples for the task to make the LM understand the task. Making a good prompt is not easy even for deep learning experts, and a service even provides a heuristic guide for manual prompt engineering based on experiences [46].

Even though automatic prompt tuning methods have proven to be effective, their adoption in Language-Model-as-a-Services (LMaaS) is currently lacking. While users can attempt to create prompts by running gradient-free prompt tuning methods using the provided LMaaS APIs, this approach poses challenges, particularly for non-experts who may find it difficult to deploy and

execute such methods in their private environments. Therefore, LMaaSs are encouraged to support automatic prompting methods in the services.

The primary obstacle to integrating both gradient-based and gradient-free prompt tuning methods into LMaaS lies in their substantial computational costs. In Table 2.2, we present the number of examples (often tens of thousands) that an LM needs to process in order to optimize a prompt for each task using various prompt engineering methods. In cases where the original papers present few-shot settings, we report the costs given 16 examples. Otherwise, we provide the actual costs as demonstrated in the papers' experiments. Considering the vast number of users that an LMaaS serves, each with unique tasks, processing such a large number of examples for a single task becomes excessively burdensome. This highlights the need for a lightweight prompt tuning method specifically tailored for LMaaS.

Among the methods listed in the table, MetaPrompting stands out as requiring relatively low computation resources. MetaPrompting focuses on finding improved initializations for prompts, facilitating rapid tuning and resulting in enhanced prompt performance. However, it should be noted that MetaPrompting relies on computing gradients, necessitating the deployment of a training system within the service, which introduces an additional significant overhead. As a result, there is a clear motivation for the development of prompt tuning methods that do not rely on gradients, particularly when LMaaS is in an online setting. Such methods would help alleviate the deployment burden associated with gradient computation while still enabling effective prompt optimization within the LMaaS framework.

2.6 Efficiency of prompt engineering methods

In this thesis, we explore the effectiveness of prompt engineering methods by evaluating their performance in terms of accurate prediction and the computational costs associated with prompt creation. The performance of accurate prediction is assessed using metrics such as accuracy or F1 score when the prompts generated by a method are applied to an LM. For the sake of simplicity, we refer to this efficiency as simply p̈erformance," aligning with the terminology commonly used in deep learning research. Additionally, we compare the computation costs involved in creating a prompt for a specific task. We quantify this by measuring the number of examples that an LM must process to generate a prompt. Although this measure provides a rough estimate of FLOPs, it enables a much more simple comparison of the costs by removing minor differences between the methods. we exclude the costs of the preparation process, such as the meta-learning phase of MetaPrompting, in order to focus on the scalability of prompt engineering methods with respect to the number of tasks.

2.7 Meta-learning

Meta-learning is a learning process to improve a learning algorithm using various training episodes, which is often called *learning-to-learn*. One of the most representative approach is learning initialization of parameters for fast learning from few-shot examples [17, 48]. Dataset distillation [64, 3] is an another example. It learns how to compress a dataset for computation efficiency while a learning on the compressed dataset still generalizes well.

Meta-learning have been defined as various inconsistent ways, but we borrow a bi-level optimization view to formally introduce meta-learning. Hospedales

et al. [25] formalized meta-learning as follows:

$$\begin{aligned} \boldsymbol{\omega}^* &= \arg\min_{\boldsymbol{\omega}} \sum_{i=1}^{M} \mathcal{L}^{meta}(\boldsymbol{\theta}^{*(i)}(\boldsymbol{\omega}), \boldsymbol{\omega}, \mathcal{D}^i_{val}) \\ s.t. \quad \boldsymbol{\theta}^{*(i)}(\boldsymbol{\omega}) &= \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{task}(\boldsymbol{\theta}, \boldsymbol{\omega}, \mathcal{D}^i_{train}), \end{aligned} \tag{2.1}$$

where $\mathcal{L}^{meta}$ refers to meta-learning and $\mathcal{L}^{task}$ refers to base-learning that is conditional on the learning strategy ω. $\boldsymbol{\omega}$ is often called meta-knowledge and could be initialization or dataset compression strategy in the examples we presented. The meta-learning learns $\boldsymbol{\omega}$ such that the model generalizes well on their validation sets after the base-learning with $\boldsymbol{\omega}$.

Chapter 3

Context Regularization for Gradient-based Prompt Tuning

3.1 Motivation

Gradient-based prompt tuning is one of the most popular automatic prompting methods, and has demonstrated promising performances on diverse NLP tasks. Even though prompting has unique advantages in small space requirements and its simplicity compared to other LM adaptation methods such as fine-tuning or LoRA, as we mentioned in Section 2.3, gradient-based prompting methods have shown degenerated performances compared to such LM adaptation methods. This performance gap motivates improving SGD-based prompt tuning methods. In this section, we propose a novel SGD-based prompt tuning scheme that can be applied to various SGD-based prompt tuning methods to improve their performances.

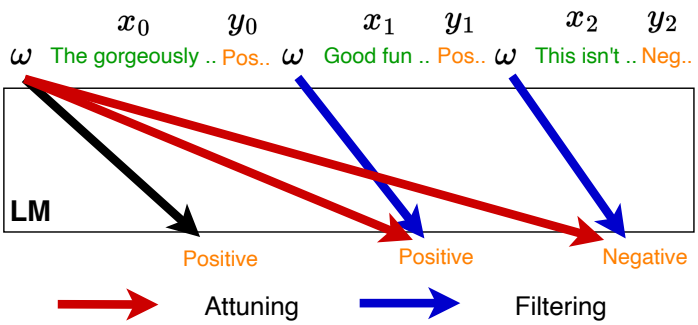

Figure 3.1 Abstracted illustration of the two regularizers, context attuning (red line) and context filtering (blue line), of CoRe with sentiment analysis data. The black line indicates regular gradient-based prompt tuning where an input sequence consists of one data example (x_0 and y_0) and the prompt ω_0 is optimized with respect to y_0 only.

3.2 Approach

Our key approach to improve SGD-based prompt tuning is to concatenate multiple examples and regularize a context generated by them, which transfers to improvement in zero-shot prediction. Our training scheme first concatenates multiple examples from the same task. Taking an example of concatenating three examples $(x_0, y_0, x_1, y_1, x_2, y_2)$ as depicted in Figure 3.1, a model is given an input $S = \{\boldsymbol{\omega}, x_0, y_0, \boldsymbol{\omega}, x_1, y_1, \boldsymbol{\omega}, x_2, y_2\}$ where $\boldsymbol{\omega}$ is the tunable prompt. Then, CoRe introduces context attuning regularizer. Context attuning optimizes the first prompt ($\boldsymbol{\omega}$ that comes before x_0) towards minimizing the losses of the succeeding examples (y_1, y_2). Therefore, the prompt does not become biased to $\{x_0, y_0\}$, but is generalized for the appended examples as well and generates less biased task contexts.

However, context attuning alone does not show the effect of regularization because the following examples receive too diverse contexts, such as styles, top-

ics, relationships between entities, or task information. Such diversity hinders the optimization of the regularizer towards the task context. To resolve this problem, we add another regularizer, context filtering, to adequately extract the task context from the preceding examples. Specifically, following the example in Figure 3.1, context filtering (blue line) optimizes prompts given preceding examples — optimization of ω_1 with respect to y_1 when $\{x_0, y_0, x_1\}$ is given.

We target zero-shot ICL where only a prompt and a test input are provided (i.e., no demonstration examples). During inference, a model is given $\boldsymbol{\omega}, x_i$ and expected to predict y_i where $\boldsymbol{\omega}$ has been trained with CoRe. We denote this setting as *zero-shot in-context learning* to imply that the model learns from the context provided by the tuned prompt, but we do not give any demonstration examples (zero-shot example for in-context learning). Please note that this differs from the typical zero-shot setting. Although the model still does not see any training example for making predictions, training data has been used when optimizing the prompt with CoRe.

We would like to highlight our finding that prompts tuned with CoRe in a few-shot ICL setting show improved performances in a zero-shot setting. CoRe's two regularizers — context attuning and context filtering— are applied on a concatenation of multiple examples (i.e., few-shot ICL), and the regularizers steer the prompt during training time to generate a more generalized task context for following examples (x_1, x_2) in a sequence. Such a well-generated task context (generated by $\boldsymbol{\omega}$) is propagated not only to the following examples (x_1, x_2) but also to the example (x_0) paired with the prompt. Therefore, a prompt tuned with CoRe helps the model's prediction when given an input of a single example $(S = \{\boldsymbol{\omega}, x_i\})$, which is equivalent to a zero-shot ICL setting.

To the best of our knowledge, this work is the first to leverage interactions between examples in a sequence for the purpose of regularization. There are

several works training on multiple examples in a sequence [47, 7, 18]. They focused on improving few-shot in-context learning by maximizing the likelihood of an example given demonstration examples. Our work is distinct from the above works in two aspects. First, we target zero-shot settings. Second, we introduce a novel regularization technique — context attuning— to improve zero-shot performance by leveraging the influence between multiple examples within a sequence.

Throughout this section, we use a simple form of prompt for conciseness of the explanation but without loss of generality: $\{\boldsymbol{\omega}, x_i, y_i\}$ where $\boldsymbol{\omega}$ is a tunable prompt. Our work focuses on autoregressive language models (LMs), so our method is not explored or analyzed upon autoencoding LMs such as BERT [14].

3.2.1 Context attuning

Context attuning prevents a prompt from being biased to a single example for generalization on the zero-shot ICL setting, using regularization on a context. In detail, this regularizer guides a prompt $\boldsymbol{\omega}$ and an example $\{x, y\}$ to create an appropriate task context without being biased towards the example.

Mechanism

To realize the goal, we first put multiple examples in an input sequence so that examples can exchange the task context with one another. This is different from typical SGD-based prompt tuning where each input sequence has only one training example, and the gradients cannot flow between the examples. In the sequence, the appended examples attend to the preceding examples and are influenced by the context of preceding examples during predictions. The prompt of the first example plays a key role in context attuning— it minimizes not only the losses of the first example (black line in Figure 3.1) but also the

losses of the succeeding examples (blue line in Figure 3.1). We hypothesize that such optimization inherently regularizes a task context from the prompt and prevents the prompt from being too biased to a single example, ultimately achieving better generalization ability.

We give a formal definition of the context attuning regularizer. In autoregressive LMs, as depicted in Figure 3.1 with the red lines, our training method optimizes as the following:

$$\boldsymbol{\omega} \longleftarrow \boldsymbol{\omega} - \epsilon \cdot \nabla_{\boldsymbol{\omega}_0} \sum_{k>0} p_{\boldsymbol{\theta}}(y_k | S_{<k}, \boldsymbol{\omega}_k, x_k), \tag{3.1}$$

where $S_k = \{\boldsymbol{\omega}_k, x_k, y_k\}$ is the k-th example in the sequence, $S_{<k} = \{S_j | j < k\}$, $\boldsymbol{\omega}_k$ is a trainable parameterized prompt, s is the number of concatenated examples, and $\boldsymbol{\theta}$ is the parameters of the LM. $\boldsymbol{\omega}_k (k \neq 0)$ is the same with $\boldsymbol{\omega}_0$ but regarded as a constant and not optimized. We introduce k to represent the positions of the prompts and indicate which prompts are optimized. On our method, the prediction of each example $S_{i,j}$ is conditioned on the prompt $\boldsymbol{\omega}_0$ paired with the first example, *"The gorgeously.."* in Figure 3.1 for an example, and the prompt is optimized for multiple examples.

We were inspired by few-shot ICL [54, 4] when designing the context attuning regularizer. When predicting the answer for a new input, few-shot ICL prepends a few *demonstrations* — input-answer pairs — to the new input and condition on those demonstrations to understand the task. The success of few-shot in-context learning has shown that prepended examples provide some meaningful information (e.g., task context) to succeeding data. Few-shot ICL leverages the interaction among examples during inference; our method can be considered as leveraging the same advantage during training (prompt tuning) to achieve better generalization.

Theoretical analysis

We analyze how the regularizer affects context attuning using a theoretical framework from Xie et al. [66] and verify that context attuning improves zero-shot inference. Xie et al. [66] designed the framework to explain what enables in-context learning of LMs. They viewed an LM as Bayesian inference with a hidden Markov model (HMM), and the transitions of the HMM are parameterized by a latent concept $\mathbf{c}$, which represents a task. On their framework, our regularizer can be expanded as follows:

$$p(y_k|S_{<k}, \boldsymbol{\omega}_k, x_k) \propto \int_{\mathbf{c}} \sum_{h_k^s} p(y_k|x_k, h_k^s, \mathbf{c}) p(h_k^s|S_{<k}, \boldsymbol{\omega}_k, x_k, \mathbf{c}) p(S_{<k}, \boldsymbol{\omega}_k, x_k|\mathbf{c}) p(\mathbf{c})\, d\mathbf{c}, \quad (3.2)$$

where h_k^s is a hidden state corresponding to x_k. Note that we omit the index of sequence i as our analysis is done on a single sequence.

We are interested in how the regularizer attunes the task context and generalizes to maximize $\sum_i p(y_{i,0}|\boldsymbol{\omega}_0, x_{i,0})$, which is zero-shot inference. On the framework, our regularizer affects the terms that $\boldsymbol{\omega}_0$ is involved in: $p(h_k^s|S_{<k}, \boldsymbol{\omega}_k, x_k, \mathbf{c})$ and $p(S_{<k}, \boldsymbol{\omega}_k, x_k|\mathbf{c})$. We analyze the two terms to identify the effect of the cross-data regularizer on zero-shot inference.

First, optimizing $p(h_k^s|S_{<k}, \boldsymbol{\omega}_k, x_k, \mathbf{c})$ prevents the prompt from being biased towards a single data instance by optimizing the task context passed to the following examples. The term we optimize can be expanded as follows:

$$p(h_k^s|S_{<k}, \boldsymbol{\omega}_k, x_k, \mathbf{c}) = \sum_{h_0^s} p(h_k^s|h_0^s, S_{<k-1}^{-\boldsymbol{\omega}_0}, \boldsymbol{\omega}_k, x_k, \mathbf{c}) p(h_0^s|\boldsymbol{\omega}_0, x_0, \mathbf{c}), \quad (3.3)$$

where $S_{<k-1}^{-\boldsymbol{\omega}_0} = S_{<k-1} - \{\boldsymbol{\omega}_0\}$. Optimizing only zero-shot inference $p(y_0|\boldsymbol{\omega}, x_0)$ may cause the hidden state h_0^s to be biased to the data instance. However, with our regularization, the prompt $\boldsymbol{\omega}_0$ is tuned to pass the proper task context

via the hidden state h_k^s to the following examples. To increase the probability of such hidden state, the prompt should be tuned to increase the probability $p(h_0^s|\boldsymbol{\omega}_0, x_0, \mathbf{c})$ where h_0^s is likely to transit to the proper hidden state h_k^s, where the first term of RHS of Equation 3.3, $p(h_k^s|h_0^s, S_{<k-1}^{-\boldsymbol{\omega}_0}, \boldsymbol{\omega}_k, x_k, \mathbf{c})$, is high. Since h_0^s is unlikely to transit the hidden states that embed the proper task context when h_0^s has already been biased to a specific data instance, the bias is naturally avoided. Finally, the unbiased hidden state h_0^s improves the average zero-shot inference performance, which is the average of Equation 3.2 over the dataset where $k = 0$.

Moreover, the optimization also calibrates $p(S_{<k}, \boldsymbol{\omega}_k, x_k|\mathbf{c})$ to align the input data to the optimal concept. Comparing that the original method only optimizes $p(\boldsymbol{\omega}_0, x_{i,0}|\mathbf{c})$, $S_{<k}$ contains y_0 so we can align the input to the task additionally considering the answer for the input.

3.2.2 Context filtering

The context filtering regularizer guides prompts to filter and receive only the task-related context (task context). With context attuning regularizer, CoRe help LMs to obtain unbiased hidden states for zero-shot inference but the context attuning regularizer alone cannot realize the regularization effect if hidden states of succeeding examples are too noisy.

Hidden states deliver various contexts, including not only task contexts but also contexts related to the style, topics, or content of preceding examples. The following examples' predictions depend on the mixed contexts. Such phenomenon has been empirically reported in the work of Liu et al. [41] where each example has a unique set of optimal demonstrations.

We conjecture that contexts that are not directly related to the task add noise to optimization on hidden states, undermining the regularization effect of

context attuning regularizer. Thus, there should be some auxiliary mechanism that selects only the set of hidden states that convey task-related optimal signal and the context filtering regularizer serves that role.

Context filtering regularizer steers a parameterized prompt to filter only the set of hidden states that convey task contexts. We maximize the likelihood of an example given prepended demonstration examples, and this is the same loss with MetaICL [47] and ICT [7] but without meta-learning. Specifically, we use the following SGD step:

$$\boldsymbol{\omega} \longleftarrow \boldsymbol{\omega} - \epsilon \cdot \sum_{k>0} \nabla_{\boldsymbol{\omega}_k} p_{\boldsymbol{\theta}}(y_k | S_{<k}, \boldsymbol{\omega}_k, x_k), \tag{3.4}$$

as depicted in Figure 3.1 with the blue lines. The only required context for inference with demonstrations is the task context extracted from the preceding demonstrations because the task is the only correlation between multiple examples. Therefore, this regularizer intrinsically leads the prompt to extract a proper task context from preceding examples.

3.2.3 Practical objectives

Our SGD step consists of the original likelihood maximization and the two regularizers:

$$\begin{aligned} \boldsymbol{\omega} \longleftarrow \boldsymbol{\omega} &- \nabla_{\boldsymbol{\omega}_0} p_{\boldsymbol{\theta}}(y_0 | \boldsymbol{\omega}_0, x_0) \\ &- \nabla_{\boldsymbol{\omega}_0} p_{\boldsymbol{\theta}}(y_1 | \boldsymbol{\omega}_0, x_0, y_0, \boldsymbol{\omega}_1, x_1) \\ &- \nabla_{\boldsymbol{\omega}_1} p_{\boldsymbol{\theta}}(y_1 | \boldsymbol{\omega}_0, x_0, y_0, \boldsymbol{\omega}_1, x_1), \end{aligned} \tag{3.5}$$

when we place two examples in a sequence. This can be implemented by simply concatenating examples and minimizing the cross entropy of all of the answers. However, when we concatenate more than two examples, we need a more complex implementation and multiple iterations for a sequence because some prompts should be regarded as constants.

For concrete analysis, we consider PyTorch-like frameworks, where a backpropagation iteration computes the gradients of a single scalar loss for listed parameters, and autoregressive LMs. On such frameworks and LMs, CoRe requires $n-1$ backpropagation iterations, where n is the number of concatenated examples. For example, when $n = 3$, CoRe needs to compute the gradient of the original likelihood G_{00}, context attuning $\{G_{01}, G_{02}\}$, and context filtering $\{G_{11}, G_{22}\}$, where $G_{ij} = \nabla_{\boldsymbol{\omega}_i} p_{\boldsymbol{\theta}}(y_j|S_{<j}, \boldsymbol{\omega}_j, x_j)$. We can compute G_{00}, G_{01}, G_{02}, and G_{22} at the first backpropagation iteration, and G_{11} at the second, resulting in two iterations. This is not scalable because we need more iterations as the number of concatenated examples increases.

For efficient training, we modify the losses by allowing $\boldsymbol{\omega}_k (k > 0)$ to be optimized for the cross-data regularizer so that we can simply compute the required gradients in a single iteration. As a result, the final SGD step of CoRe is as follows:

$$\boldsymbol{\omega} \longleftarrow \boldsymbol{\omega} - \sum_{i<n} \sum_{j<i} G_{ij}, \tag{3.6}$$

where n is the number of concatenated examples.

The modified CoRe step is similar to the combination of the original CoRe step with various number of concatenated examples. If we assume $G_{ii} \approx \nabla_{\boldsymbol{\omega}_i} p_{\boldsymbol{\theta}}(y_i|\boldsymbol{\omega}_i, x_i)$, the set of the gradients of the new CoRe objective is the same as the union of the original CoRe gradients on n inputs — $\{\boldsymbol{\omega}_i, x_i, y_i | i < j\}$ where $j \leq n$ — before the modification. For example, when $n = 3$, the original CoRe gradients on $\{\boldsymbol{\omega}_0, x_0, y_0, \boldsymbol{\omega}_1, x_1, y_1, \boldsymbol{\omega}_2, x_2, y_2\}$ are the same with $\{G_{00}, G_{01}, G_{02}, G_{11}, G_{22}\}$, those on $\{\boldsymbol{\omega}_1, x_1, y_1, \boldsymbol{\omega}_2, x_2, y_2\}$ are the same with $\{G_{11}, G_{12}, G_{22}\}$, and those on $\{\boldsymbol{\omega}_2, x_2, y_2\}$ are the same with $\{G_{22}\}$ under the assumption. Therefore, the gradient sum of the three steps are as follows:

$$\sum_{i<n} \sum_{j<i} G_{ij} + G_{11} + 2G_{22} \tag{3.7}$$

Suppressing the magnitude of G_{11} and G_{22} by multiplying 1/2 and 1/3 respectively, Equation 3.7 become the modified CoRe gradients. Therefore, the modified CoRe gradients are the same with the summation of the original CoRe gradients on various sequence lengths with some calibrations.

3.3 Experimental Setup

In this section, we briefly specify the setup that our experiments are conducted on: models, datasets, and hyperparameters.

Model and Dataset

We evaluate our training method on two large language models, GPT2-XL (1.5B parameters) and GPT-J (6B parameters), and seven classification datasets (CB [12], RTE, WSC, WiC [51], COPA [57], BoolQ [9], MultiRC [33]) from SuperGLUE benchmark [63]. The benchmark has English datasets from various tasks and domains such as news, blogs, or encyclopedia. We downloaded the model checkpoints from Huggingface transformers [65] and datasets from Huggingface datasets [38].

Baselines

We evaluate our method on three gradient-based prompt-tuning baselines: P-tuning [42], Prefix-tuning [39], and Softprompt [35].

Input Construction for CoRe

In our experiments, the input is formed by simply adding several trainable prompt embeddings in front of each element of a data instance. This way of constructing an input is highly convenient because it does not require manual

human effort and additional tuning for each task. For example, a single data instance of natural language inference tasks consists of three elements: premise, hypothesis, and label. We transform the data instance into (e_0, *premise*, e_1, *hypothesis*, e_2, *label*) where $e_i \in \mathbb{R}^{n \times h}$ is a parameterized prompt that we optimize, and h is the hidden state size. We use $n = 1$ for P-tuning and Soft-prompt. For Prefix-tuning, we use a different template since prefix-tuning appends prompts only at the front of a data instance. We transform the data instance into $(e_0, premise, hypothesis, label)$ where e_i is a parameterized prompt of size 5.

Sequence Size and Batch Size

We introduce a special hyperparameter used in CoRe, named *sequence size*. This refers to the number of data instances concatenated for a single training example when CoRe. For example, an input for CoRe of sequence size 2 and 3 becomes $\{\boldsymbol{\omega}_0, x_0, y_0, \boldsymbol{\omega}_1, x_1, y_1\}$ and $\{\boldsymbol{\omega}_0, x_0, y_0, \boldsymbol{\omega}_1, x_1, y_1\, \boldsymbol{\omega}_2, x_2, y_2\}$, respectively. Please note that CoRe of sequence size 1 is equivalent to the standard SGD training. To make a fair comparison, we keep the size of batch size = (number of sequences in a batch) $\times$ (sequence size) to be a constant. This way, a model sees an equal amount of data for every iteration of CoRe across all sequence sizes.

Evaluation

We use accuracy averaged over different random seeds. We use ten seeds for GPT2-XL experiments, and five for GPT-J experiments except for MultiRC experiments, where we use five and three seeds respectively.

Hyperparameters We search for the best set of hyperparameters (learning rate and batch size) on the search space presented in Table 3.1. We use Super-

Hyperparameter	Method	GPT2-XL	GPT-J
LR	P-tuning	$\{2e{-}4, 4e{-}4, 8e{-}4\}$	$\{1e{-}4, 2e{-}4, 4e{-}4, 8e{-}4\}$
	Softprompt	$\{2e{-}3, 2e{-}2, 2e{-}1\}$	$\{2e{-}3, 1e{-}2, 5e{-}2\}$
	Prefix	$\{8e{-}6, 4e{-}5, 2e{-}4\}$	$\{2e{-}5, 1e{-}4, 2e{-}4, 4e{-}4\}$
Batch size	All	$\{32, 64\}$	$\{32, 64\}$

Table 3.1 Hyperparameter search space for P-tuning, Softprompt, and Prefix-tuning at GPT2-XL and GPT-J

Method	GPT2-XL	GPT-J
P-tuning	4e−4	2e−4
+ CoRe	4e−4	2e−4
Softprompt	2e−2	1e−2
+ CoRe	2e−2	1e−2
Prefix	2e−4	1e−4
+ CoRe	2e−4	2e−4

Table 3.2 Learning rate for P-tuning, Softprompt, and Prefix-tuning at GPT2-XL and GPT-J

GLUE CB, the smallest dataset among SuperGLUE subsets, for quick hyperparameter search. For each search space of three methods and two models, we train a prompt using the baseline and the baseline with CoRe of sequence size 2.

We use batch size 32 for all three methods at GPT2-XL and GPT-J. Note that the batch size equals (number of sequences in a batch) $\times$ (sequence size), as we described in Section 3.3.

In Table 3.2, we report learning rate selected for each method at GPT2-XL

and GPT-J. Both the baseline and the baseline with CoRe show the best performance at the same learning rate except for Prefix-tuning at GPT-J. Therefore, for this specific case, we use the optimal learning rates for each setting (1e−4 for baseline and 2e−4 for CoRe). We use the same learning rate and batch size for all datasets.

We train prompts for 30 epochs on SuperGLUE CB, WSC, and COPA, which are small datasets, and 20 epochs on SuperGLUE RTE and WiC, which are large datasets. The size of SuperGLUE BoolQ (9K) and MultiRC (27K) are fairly larger than other SuperGLUE subsets (¡ 5K), so we match the number of training iterations for those two datasets to that of SuperGLUE RTE instead of setting epochs for them.

3.4 Experimental Results

We evaluate CoRe on the setup we presented in Section 3.3. We first show the performance gain of our method on the three baselines. Then, we present how the performance changes according to sequence sizes, and further analyze the effect of two regularizers of CoRe with the ablation studies. In addition, we present how the similarity of concatenated examples affects performance. Finally, we present the performance of CoRe in few-shot inference settings.

3.4.1 Main result

Model	Method	CB	RTE	WSC	WiC	COPA	BoolQ	MultiRC
GPT2-XL	P-tuning	76.79	68.84	**64.23**	63.01	57.60	70.90	**67.64**
	+ CoRe	**81.79**	**72.13**	**64.23**	**66.91**	**58.80**	**71.75**	65.52
	Softprompt	73.57	68.66	63.65	63.90	**60.50**	68.01	**70.48**
	+ CoRe	**82.32**	**73.50**	**64.42**	**64.95**	58.90	**71.77**	69.87
	Prefix	65.90	59.35	**63.56**	53.12	**57.70**	62.56	**72.24**
	+ CoRe	**79.46**	**61.66**	64.14	**53.64**	56.40	**67.40**	69.58
GPT-J	P-tuning	94.64	79.93	65.19	69.03	**70.20**	78.36	**84.45**
	+ CoRe	**97.50**	**84.98**	**65.77**	**70.85**	67.20	**84.01**	84.12
	Softprompt	91.07	82.24	65.00	**67.43**	61.80	82.31	**82.26**
	+ CoRe	**95.36**	**83.25**	**65.19**	67.09	**62.60**	**82.50**	80.27
	Prefix	94.20	81.88	65.19	**70.50**	**66.60**	**83.38**	**84.21**
	+ CoRe	**94.64**	**82.46**	**65.77**	66.96	65.20	82.81	81.61

Table 3.3 Comparison of zero-shot evaluation results between three baselines and applying CoRe to the baselines across various NLU datasets. For evaluation metric, we used averaged accuracy. We highlight the better one among a baseline and that with CoRe.

We first experiment with our method on the three baselines, which uses parameterized continuous prompts. Table 3.3 compares the zero-shot performances of prompts trained only with baseline tuning methods against prompts trained with CoRe on top of the methods. On GPT2-XL, CoRe shows improvements in accuracy compared to P-tuning, Prefix-tuning, and Softprompt, up to 11.9%. On GPT-J, CoRe also shows consistent enhancements up to 6.3% for the three methods. The absolute gains are reduced on GPT-J, compared to the gains on GPT2-XL. This is expected as it is typically hard to earn a large gain as the baseline performance increases.

Interestingly, we found that our method shows a higher accuracy gain for NLI tasks — SuperGLUE CB and RTE — on GPT2-XL across all three baselines. We can hypothesize that there are some correlations between NLI tasks and the pretraining objective, or between the tasks and the pretraining dataset of GPT2-XL. It is worthy to analyze the relationship between the pretraining and downstream tasks to further improve the gains for other datasets as a future work.

We observe that there are mainly two cases where CoRe does not show performance gain. First, CoRe performs worse than the baseline on SuperGLUE MultiRC. We conjecture that task context does not get properly propagated since training samples of MultiRC are far longer than samples of other SuperGLUE subsets.

Another observation is that CoRe generally does not work well for Prefix-tuning on GPT-J. We hypothesize that this might have some correlation with Rotary Positional Embedding (RoPE) of GPT-J. It has not yet been explored how the mechanism of Prefix-tuning (appending prompts to all the key and values of each layer) interact with RoPE and this may undermine proper working of CoRe.

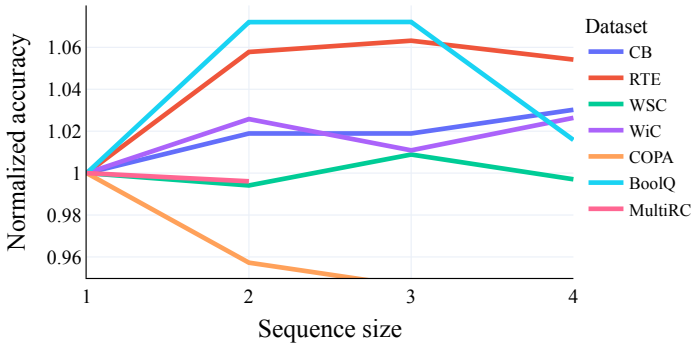

Figure 3.2 Zero-shot evaluation on varying sequence sizes. Accuracy for each dataset is normalized with respect to its accuracy when the sequence size is 1.

3.4.2 How many examples to concat?

To see how the sequence size of CoRe affects the performance, we train prompts on GPT2-XL and GPT-J using varying sequence sizes from one to four. Figure 3.2 shows the result of P-tuning on GPT-J with zero shot evaluation, and Table 3.4, Table 3.5, and Table 3.6 presents the results of all of the baselines on both models with the exact numbers.

In most cases, sequence size of 3 or 4 exhibits the best performances on GPT-J, but CoRe shows better performance with smaller sequence sizes, 2 or 3, on GPT2-XL. This shows that context attuning leads to context with less bias when given more data in a sequence, and provides effective regularization. Increasing the sequence size over a certain level results in accuracy drop. From this we infer that too many examples in a single sequence leads to causing a noise to become part of the context.

Model	s	SuperGLUE						
		CB	RTE	WSC	WiC	COPA	BoolQ	MultiRC
GPT2-XL	1 (P-tuning)	76.79	68.84	**64.23**	63.01	57.60	70.90	**67.64**
	2	**81.79**	**72.13**	**64.23**	65.03	58.70	70.16	65.52
	3	77.50	63.54	64.04	**66.91**	58.30	71.02	–
	4	78.39	71.95	63.94	63.43	**58.80**	**71.75**	–
GPT-J	1 (P-tuning)	94.64	79.93	65.19	69.03	**70.20**	78.36	**84.45**
	2	96.43	84.55	64.81	70.81	67.20	**84.01**	84.12
	3	96.43	**84.98**	**65.77**	69.78	66.40	**84.01**	–
	4	**97.50**	84.26	65.00	**70.85**	62.20	79.60	–

Table 3.4 The performance of P-tuning ($s = 1$) and CoRe on P-tuning where s is the sequence size. We highlight the best performance among all sequence sizes.

Model	s	SuperGLUE						
		CB	RTE	WSC	WiC	COPA	BoolQ	MultiRC
GPT2-XL	1 (Soft)	73.57	68.66	63.65	63.90	**60.50**	68.01	70.48
	2	75.89	73.14	**64.42**	64.80	58.20	70.31	69.87
	3	78.39	71.88	63.85	64.11	57.90	**71.77**	–
	4	**82.32**	**73.50**	63.85	**64.95**	58.90	70.74	–
GPT-J	1 (Soft)	91.07	82.24	65.00	**67.43**	61.80	82.31	**82.26**
	2	**95.36**	**83.25**	65.00	67.09	62.00	**82.50**	80.27
	3	**95.36**	83.11	**65.19**	65.36	**62.60**	82.34	–
	4	**95.36**	82.89	64.42	66.21	61.00	81.26	–

Table 3.5 The performance of Softprompt ($s = 1$) and CoRe on Softprompt where s is the sequence size. We highlight the best performance among all sequence sizes.

Model	s	SuperGLUE						
		CB	RTE	WSC	WiC	COPA	BoolQ	MultiRC
GPT2-XL	1 (Prefix)	65.89	59.35	63.56	53.12	**57.70**	62.56	**72.24**
	2	**79.46**	**61.66**	**64.14**	**53.64**	56.40	**67.40**	69.58
GPT-J	1 (Prefix)	94.20	81.88	**65.19**	**70.50**	**66.60**	**83.38**	**84.21**
	2	**94.64**	**82.46**	65.00	66.96	65.20	82.81	81.61

Table 3.6 The performance of Prefix-tuning ($s = 1$) and CoRe on Prefix-tuning where s is the sequence size. We highlight the best performance among all sequence sizes.

3.4.3 The impact of each regularizer

To see how context attuning and context filtering each contributes to the performance gain from CoRe, we conduct ablation studies with gradients involved in CoRe. We factorize the gradients as in Equation 3.5 with a sequence size of 2, and the gradients represent the gradient in the baseline method, context attuning, and context filtering, respectively. We then test the effect of each type of gradients.

As presented in Table 3.7, in general, CoRe exhibits the best performance when all three gradients (baseline, context attuning, and context filtering) are involved. Context attuning alone has no gain, or shows smaller gain than applying both regularizers. It requires a mechanism to filter unnecessary contexts and make context attuning focus on the task context, and our context filtering serves that role. There is one unexpected behavior in SoftPrompt on CB that applying the filtering gradient alone shows performance on par with CoRe. We hypothesize that filtering appropriate task context alone shows good enough performance for this specific case.

Method	Task	orig.	+ A	+ F	+ both (CoRe)
P-tuning	RTE	68.84	69.86	68.48	**72.13**
	CB	76.79	76.07	73.57	**81.79**
Softprompt	RTE	68.66	69.31	69.71	**73.14**
	CB	73.57	75.00	**75.89**	**75.89**
Prefix	RTE	59.35	55.56	53.72	**61.66**
	CB	65.90	73.39	71.07	**79.46**

Table 3.7 Ablations studies on the effect of context attuning regularizer (A) and context filtering regularizer (F) where sequence size is 2.

3.4.4 Measurement of bias caused by prompts

To showcase the regularization effect of CoRe, we compare how much prompts are biased to training data samples before and after applying CoRe. For quantifying the amount of bias, we use 'one-step generalization ratio (OSGR)' suggested by [40], which is a validation loss drop during a single training step divided by a training loss drop during the step. We calculate the training loss only on the single batch of data used in the training step, not the entire training set, to represent how much the prompt is over-fitted (biased) to the batch. The higher the OSGR, the faster the validation loss drops with respect to the training loss, meaning smaller generalization gap between a training batch and a validation set. Note that a model sees the exactly same training data in a single training step regardless of applying CoRe, as we mentioned in Section 3.3. In Table 3.8, we report results for three baselines on CB and RTE datasets, measured for 100 training steps after 3 epochs, and averaged over 10 seeds.

Despite training for the same batch of data examples, all three baselines

Method	Task	orig.	+ CoRe
P-tuning	RTE	0.024	**0.050**
	CB	0.261	**0.286**
Softprompt	RTE	0.006	**0.014**
	CB	0.137	**0.157**
Prefix	RTE	0.013	**0.020**
	CB	0.015	**0.037**

Table 3.8 Comparison of one-step generalization ratio [40] before and after applying CoRe.

Similarity	CB	RTE	WiC
max	78.93	67.22	62.79
max (10%)	76.96	70.43	64.70
standard	**81.79**	**72.13**	**65.03**
min (10%)	77.68	62.13	64.67
min	81.07	69.35	64.48

Table 3.9 Comparison of zero-shot performance of various sampling methods considering semantic similarity. Standard is CoRe without such sampling.

show a smaller OSGR than the baselines with CoRe. This result implies that vanilla prompt-tuning is prone to make a prompt more biased toward data examples that the model has seen during training. CoRe produce prompts that can mitigate such bias which aligns with the performance gain shown in Table 3.3.

3.4.5 Which examples to concat?

Previous studies show that concatenating semantically similar examples improves downstream task performances. Liu et al. [41] empirically shows that

demonstration examples that are semantically similar to the target example improve in-context learning performances. Some studies also show that placing semantically similar examples in an input sequence during finetuning [18] or pretraining [37] improves downstream task performance. We experiment how semantic similarity between examples in an input sequence affects the performance of CoRe. We consider CoRe with sequence size 2.

For the experiment, we first sample *batch size*/2 samples for each iteration during prompt tuning. For each sampled example, we select the most similar example (*max*) or the least similar example (*min*) among the unseen examples in the epoch, and append the selected example after the sampled example. We also experiment with less extreme similarity by sampling an example to be appended from 10% most similar (*max 10%*) or 10% least similar (*min 10%*) examples of the unseen examples in the epoch. We measure the semantic similarity between examples using stsb-roberta-large model from sentence transformers [56].

As presented in Table 3.9, sampling considering semantic similarity degenerates the performances of CoRe compared to the standard sampling, where demonstrations are sampled at random. This shows that CoRe also benefits from concatenating semantically similar examples — standard sampling reports better evaluation result compared to *min* and *min 10%*. However, choosing to concatenate the more similar example (*max* and *max 10%*) introduces a bias to the prompt, and the bias depresses the regularization effects of CoRe.

3.4.6 Few-shot in-context learning

We also evaluate our method in a few-shot in-context learning setting, where we prepend multiple examples as demonstrations to a target example during evaluation. Although the few-shot setting is not originally our target, the objective of two regularizers — to optimize a prompt towards sending and receiving

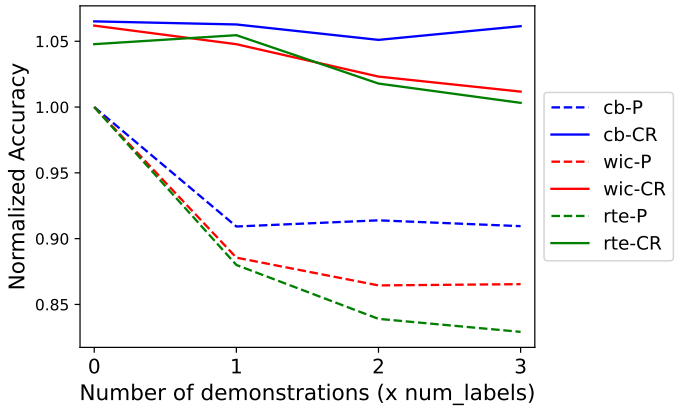

Figure 3.3 Comparison of few-shot in-context learning performance between P-tuning and P-tuning with CoRe, normalized by zero-shot performance of P-tuning. Number of demonstrations we use is a multiple (x axis) of the number of each dataset's classes.

a proper task context — naturally contributes to improving the predictions prepended with demonstrations — that is, few-shot in-context learning. Figure 3.3 shows the result of few-shot evaluation on three different datasets with prompts from P-tuning and CoRe on P-tuning.

We observe that both P-tuning and CoRe on P-tuning report degradation in general as the number of demonstrations increases. However, CoRe on P-tuning exhibits far less accuracy drop across different numbers of demonstrations and even performance gain in the case of CB dataset. We assume that less degradation compared to P-tuning comes from two regularizers of CoRe.

3.5 Discussion

In this section, we discuss the limitations of CoRe, and suggest future research directions to resolve them.

No improvement on some tasks

As reported in Section 3.4.1, NLI tasks demonstrate a significant benefit from CoRe, while other tasks show marginal benefits or no benefits at all. We hypothesize that this disparity stems from the specific characteristics of each task. However, we have not yet conducted a comprehensive investigation to identify the task characteristics that contribute to the observed performance gains.

To address this issue, we require a novel deep learning interpretation method that can effectively probe the latent contexts of LM. Alternatively, a thorough analysis of the relationship between the pretraining objective and downstream tasks, as well as an examination of how prompting bridges the gap between these two distinct phases, would provide valuable insights. We consider these research questions as part of our future work, awaiting further exploration and investigation.

Long texts

CoRe is not suitable for cases where the training dataset consists of excessively long sequence texts. This is because CoRe requires the concatenation of multiple examples, making it impractical for developers to benefit from CoRe if the majority of concatenated examples from their dataset exceed the maximum sequence length of the language model. It is important to acknowledge, however, that recent advancements in language models have begun to address this limitation by adopting longer sequence sizes. The inclusion of longer sequence sizes enables CoRe to handle longer examples more effectively, thereby mitigating this constraint.

Generation tasks

We have yet to examine the applicability of CoRe to natural language generation (NLG) tasks. NLG holds a significant position in natural language processing research, alongside natural language understanding (NLU), and presents numerous intriguing applications. We are convinced that the concept of context attuning and context filtering, central to CoRe, can provide assistance in addressing major challenges in NLG, such as controlled NLG. Following the submission of this paper, our future plans involve delving into the exploration of CoRe on NLG tasks.

Chapter 4

Meta-Learning of Prompt Generation for In-context Learning

4.1 Motivation

As discussed in Section 2.5, despite the effectiveness of automatic prompt tuning methods and their potential benefits for non-expert users of Language-Model-as-a-Services (LMaaS), their adoption in LMaaS is currently lacking. The main hurdle in incorporating both gradient-based and gradient-free prompt tuning methods into LMaaS stems from their significant computational costs. Moreover, some methods necessitate the deployment of training systems for gradient-based prompt engineering, which further adds to the overhead and challenges of implementing these methods in LMaaS.

4.2 MetaL-Prompt

To address the challenges associated with automatic prompt tuning in LMaaS, we introduce MetaL-Prompt, a meta-learning approach for a lightweight prompt generation. In this approach, we meta-train a prompt generation model (PGM) (Section 4.2.1), which is initialized with a target language model (LM), with the objective of generating prompts that enhance the LM's contextual learning capabilities. We refer to this training process as meta-learning because the PGM learns generation of prompts that effectively induce meaningful contexts for the target LM to learn from (i.e., learning-to-learn). We also propose the use of trainable padding (Section 4.2.2) to alleviate the overhead of the prompt generation process, which requires multiple forward passes, during the meta-learning. Additionally, we explore various types of prompts that PGMs can generate, as discussed in Section 4.2.3, which have not been explored in previous prompt generation methods.

The overall workflow of an LMaaS with MetaL-Prompt is depicted in Figure 4.1. Initially, the model provider trains a PGM using MetaL-Prompt (Figure 4.1 (a)). Importantly, this training process does not impact the ongoing service as the training occurs prior to the service initiation. During the service, when a user provides a set of few-shot examples for the user's specific task, the prompt generation model generates a prompt based on these few-shot examples (Figure 4.1 (b)). This generated prompt is optionally further tuned with gradient-based prompt tuning for improved performance. The final prompt is then saved and associated with the user's future requests. Finally, when the user submits a request pertaining to the task, the prompt is composed to the query, and the composed input is fed into the LM to generate the response (Figure 4.1 (c)).

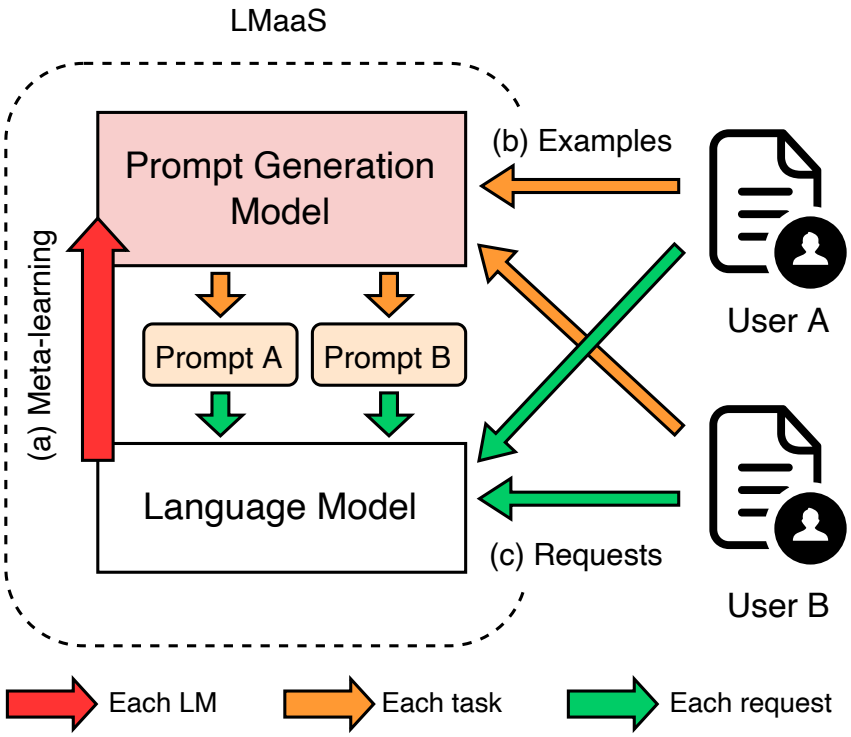

Figure 4.1 A workflow of MetaL-Prompt on LMaaS.

MetaL-Prompt offers advancements over prior works in two key aspects: prompt quality and computation cost for prompt generation. The prompts that MetaL-Prompt generate empirically demonstrate more accurate or comparable prediction quality compared to prompts crafted by previous gradient-based or gradient-free approaches [42, 36, 61, 13], given limited computation budgets. In terms of computational efficiency, MetaL-Prompt surpasses previous methods by requiring only a single forward pass for prompt generation, as explained in Section 4.2.2. This streamlined approach is highly productive, especially when contrasted with prior approaches that necessitate an extensive number of forward or backward passes to tune or generate a prompt for

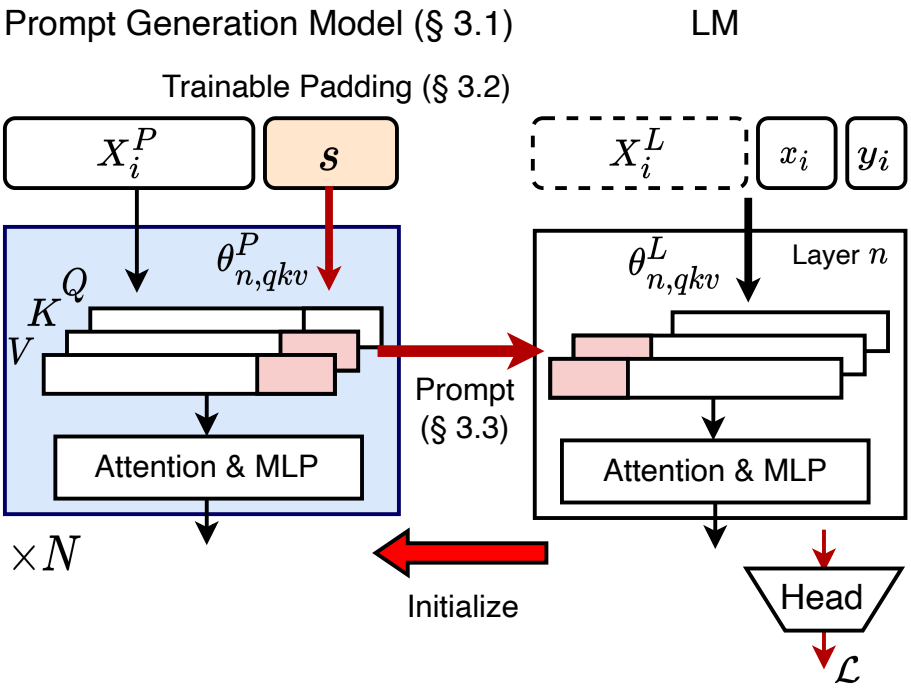

Figure 4.2 An illustration of meta-learning of a prompt generation model.

a single task, as discussed in Section 4.1. The reduction in computation cost achieved by MetaL-Prompt significantly enhances its practicality and efficiency with LMaaS. Furthermore, MetaL-Prompt does not necessarily rely on gradient computation during prompt generation, thereby reducing the deployment overhead of training systems. Since MetaL-Prompt still requires gradients during the meta-learning phase of a PGM, MetaL-Prompt is a hybrid approach in terms of gradient usage, as described in Table 2.1.

4.2.1 Prompt generation model (PGM)

MetaL-Prompt employs meta-training to train a prompt generation model (PGM), enabling the creation of prompts that enhance in-context learning of the target

language model (LM) across diverse tasks. The objective function utilized by MetaL-Prompt, as depicted in Figure 4.2, can be expressed as follows:

$$\boldsymbol{\theta}_P^* = \arg\max_{\boldsymbol{\theta}_P} \sum_i p(y_i | f_{\boldsymbol{\theta}_P}(X_i^P), X_i^L, x_i; \boldsymbol{\theta}_L) \tag{4.1}$$

where $\boldsymbol{\theta}_L$ and $\boldsymbol{\theta}_P$ are the parameters of the LM and the PGM respectively, $X_i^P = \{x_{i,0}^P, y_{i,0}^P, x_{i,1}^P, y_{i,1}^P, ...\}$ and $X_i^L = \{x_{i,0}^L, y_{i,0}^L, x_{i,1}^L, y_{i,1}^L, ...\}$ are concatenations of examples. X^P and X^L consist of examples from various NLP tasks. The prompt generation process $f_{\boldsymbol{\theta}_P}$ can be realized in different ways, and one straightforward example is choosing the most probable next tokens using probabilities predicted by the PGM. Further details regarding the prompt generation process will be discussed in-depth in Section 4.2.3.

The parameter $\boldsymbol{\theta}_P$ is initialized with a target LM to leverage its existing understanding of various NLP tasks. In essence, MetaL-Prompt employs Equation 4.1 to adapt the LM and obtain a PGM. For this adaptation process, we utilize LoRA [29], a parameter-efficient fine-tuning method, instead of full-parameter fine-tuning. Note that the original LM is frozen and only PGM is tuned during the meta-learning.

During the service phase following meta-learning, when a user provides a set of few-shot examples, MetaL-Prompt divides them into two subsets: X^P and X^L. The PGM utilizes X^P to generate an appropriate prompt. This generated prompt is then combined with the additional demonstration examples X^L, and the composed input is used to process future user requests. It is important to note that X^L can be an empty set. In such cases, only the prompt and the input from the user's request are fed to the LM. This configuration enables the fastest inference speed due to the shorter sequence lengths.

A prompt generated by the PGM can be directly utilized for LM prompting. However, for cases where additional computational cost is acceptable, we can

enhance the prompt by applying gradient-based prompt tuning methods [36, 42, 39, 60]. In this scenario, the generated prompt serves as an initialization of gradient-based prompt tuning. We observe that this additional tuning process can further improve the performance of the prompt.

4.2.2 Trainable padding

Generative language models, such as GPT-3 [4], typically predict one token at a time within a given context, necessitating n forward passes to generate n tokens. This multi-pass generation process can not only introduce inefficiencies in existing serving systems [68] but also cause extra overheads to train PGMs with the objective outlined in Equation 4.1, which also includes generation processes.

We tackle this challenge by proposing trainable padding, which is inspired by special tokens of recent LMs and gradient-based prompt tuning [36, 42]. As depicted in Figure 4.2, MetaL-Prompt appends trainable embeddings to the given examples X^p, which are then fed to the PGM as part of the input. This enables the PGM to generate multiple prompt tokens simultaneously by leveraging the hidden states corresponding to each padding position. Additionally, we reparameterize the trainable padding similar to other prompt tuning methods [42, 39, 43, 27], specifically employing LSTMs following the methodology of P-tuning [42].

4.2.3 Prompt design

In this section, we discuss four prompt designs — *Discrete*, *Weighted Sum*, *Hidden State*, and *Prefix* — and their generation using a PGM. While existing prompt generation methods have primarily focused on discrete prompts in natural language, we extend our investigation to include real-valued prompts

(i.e., continuous prompts) as previous prompt tuning methods [42, 39, 61] have demonstrated their effectiveness.

Discrete is a prompt that consists of the most probable natural language tokens predicted by a PGM. To train the PGM for Discrete, we adopt Gumbel-Softmax reparameterization with discretization ("Straight-through" trick).

Weighted Sum is a continuous prompt obtained by multiplying the token probability predicted by a PGM with the word embeddings of a target LM. It represents the probability-weighted sum of the word embeddings.

Hidden State directly uses the input hidden states of the head layer in a PGM as a continuous prompt. As word embedding layers and head layers typically share the same parameters in recent LMs, the input hidden states from the head layer have the same representations as the word embeddings. This makes them valid inputs (i.e., prompts) for the LM.

Prefix adds a prompt before the keys and values of transformers [39], rather than prepends it to the inputs like the others. As depicted in Figure 4.2, we extract the keys and values of self-attention layers from each layer of a PGM at the position of the trainable padding, and prepend them to the keys and values of a target LM in the corresponding layers. Prefix demonstrates the best performance among all (Section 4.4.4), and therefore, we utilize Prefix in the following experiments (Section 4.4).

4.3 Experimental setup

In this section, we present the setup of our experiments for MetaL-Prompt, which includes datasets, training and evaluation details, baselines, and models.

Setting	Split	
	Meta-learning	Eval
cls→cls	43	20
HR→LR	61	26
QA→QA	37	22

Table 4.1 The number of datasets for each task setting. There is no overlap between the meta-learning datasets and the evaluation datasets in each setting.

Dataset

We conduct experiments to evaluate the performance of MetaL-Prompt using the combination of CrossFIT [67] and UNIFIED QA [34], which consists of 142 diverse datasets. Specifically, we adopt three different task settings from MetaICL [47], $cls \rightarrow cls$, $HR \rightarrow LR$, and $QA \rightarrow QA$. Each setting defines two disjoint sets — meta-learning datasets and evaluation datasets — and brief explanations of the settings are as follows.

Classification to classification (cls → cls): Both the meta-learning datasets and the evaluation datasets encompass classification tasks.

QA to QA (QA → QA): Both the meta-learning datasets and the evaluation datasets consist of Question-Answering (QA) tasks.

High Resource → Low Resource (HR → LR): In this particular setting, meta-learning datasets comprises datasets that include more than 10,000 training examples, while each evaluation dataset consists of datasets with fewer than 10,000 examples.

These settings cover a total of 133 unique tasks, which is significantly larger than the number of tasks explored in previous prompt tuning methods [61, 62, 13, 52, 24, 72]. The statistics of each setting are described in Table 4.1.

Evaluation

We use Macro-F1, which is a better measure than accuracy on imbalanced datasets, as our evaluation metric and report the mean scores obtained from four distinct runs. For each run, MetaL-Prompt and the baselines are given a different set of 16 examples sampled from the training split of the evaluation dataset for prompt generation or tuning. MetaL-Prompt trains only a single PGM for all of the runs, but the PGM generates unique prompts for each run by leveraging distinct example sets. Furthermore, we explore the impact of varying example sizes and provide the corresponding results. Here, n_P denotes the number of examples used for prompt generation, and n_L represents the number of examples utilized for in-context learning demonstrations in addition to the tuned or generated prompts.

Training

MetaL-Prompt trains a PGM on the meta-learning datasets of each setting. Subsequently, for evaluation, the trained PGM is employed to generate prompts from n_P examples of the unseen evaluation datasets. As we mentioned in Section 4.2.1, the generated prompts can be further tuned with SGD as the previous gradient-based tuning methods. If not specified, we do not adopt the additional tuning.

As described in Section 4.2.1, MetaL-Prompt concatenates examples to form X_i^P for prompt generation and X_i^L for inference, respectively. If not mentioned, we use the same n_P and n_L with the evaluation settings to construct X_i^P and X_i^L for alignment between the training and evaluation settings.

To train the prompt generation models (PGMs) of MetaL-Prompt, we utilize the AdamW optimizer [44] along with linear learning rate decay. The learning

rate is initialized at 0.0001 for HR→LR and 0.0002 for cls→cls and QA→QA, without employing a learning rate warmup. The training epochs for cls→cls, HR→LR, and QA→QA are set to 10, 6, and 8, respectively. The prompt length for MetaL-Prompt is 20. For the additional tuning of the generated prompts, we employ an initial learning rate of 0.02 and tune them for 9 epochs to align with the computation costs associated with the gradient-based prompt tuning baselines (Section 4.3). It is important to note that this additional tuning process is performed on the few-shot examples utilized by the PGM for prompt generation.

Baselines

We compare MetaL-Prompt against five different prompt tuning baselines — P-tuning [42], SoftPrompt [36], Prefix-tuning [39], RLPrompt [13], and BBTv2 [61]. In order to tune prompt with the baselines, we use the hyperparameters listed in Table 4.2.

For P-tuning [42], SoftPrompt [36] and Prefix-tuning [39], we tune learning rates based on F1 scores of three classification tasks — AG News [20], Yelp Polarity [70] and TabFact [6] — that have the largest test splits among classification datasets, and we borrow other hyperparameters from the original paper. We set a prompt length of 20.

To train a policy model for RLPrompt [13], we adhere to the hyperparameters described in the original paper. In our experiments, we employ distilGPT-2 [30, 58], as a policy model for generating optimized prompts following the original paper. To maintain consistency with the recommendations of Deng et al. [13], we set the prompt length to 5.

Given that BBTv2 [61] leverages the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [21, 22] during its training, we adopt all hyperpa-

Method	P-tuning	SoftPrompt	Prefix-tuning	BBTv2	RLPrompt
Optimizer	AdamW	AdamW	AdamW	-	Adam
Learning Rate	1.6e−4	4e−5	2e−4	-	5e−5
Learning Rate Schedule	Linear	Linear	Linear	-	Constant
# Epoch	10	10	10	240	5
Batch Size			16		
Prompt Length	20	20	20	50	5

Table 4.2 Hyperparameters for the baselines.

rameters for CMA-ES as outlined by Sun et al. [61], with the exception of the population size and a budget to limit the computation cost. In line with Sun et al. [61], we set the prompt length to 50.

Models

To evaluate MetaL-Prompt and the baselines, we employ autoregressive LMs: GPT2-Large (762M parameters) and GPT2-XL (1.5B parameters) [54]. The motivation behind the model choice is that autoregressive models are widely utilized for LMaaS, such as OpenAI API. However, MetaL-Prompt is not limited to such models. It can support other LMs such as sequence-to-sequence models as well.

Computation costs

To assess the effectiveness of the baselines on LMaaS, we evaluate their performance under constrained cost settings, roughly 10 epochs of forward and backward passes. When MetaL-Prompt adopts the further tuning of the generated prompts, we run 9 epochs of the tuning to meet the computation budget, considering the prompt generation costs. Table 4.3

Method	forward	backward
P-tuning	160	160
SoftPrompt	160	160
Prefix-tuning	160	160
RLPrompt	320 $\times\alpha$	-
BBTv2	3,880	-

Table 4.3 Number of examples that the baselines process with the forward and backward passes for each task.

For P-tuning, SoftPrompt and Prefix-tuning, we tune prompts with the baselines for 10 epochs. It implies that they process 160 examples (10 epochs $\times$ 16 examples) with their forward and backward passes.

In accordance with the cost limit, RLPrompt is trained for 5 epochs. In our setting, RLPrompt evaluate the losses of four prompts for each example, requiring 64 forward passes per epoch. Additionally, RLPrompt also requires the losses of all classes and some classes are mapped to multi-token labels, which multiplies the required passes by the number of classes (α). For example, if a task has two classes, RLPrompt requires 128 forward passes on the setting. It is important to note that we do not consider backward passes of the policy model, as it solely updates small MLP layers [13] just before the head layer.

To assess BBTv2 in a constrained budget scenario, we designate 240 forward passes per batch for GPT2-XL and 180 forward passes for GPT2-Large, taking into account the number of hidden layers in each model (e.g., 48 layers for GPT2-XL) and the selected population size of 5. With a batch size of 16 utilized by BBTv2, this translates to a total of 3,840 forward passes for GPT2-XL and 2,880 forward passes for GPT2-Large.

Model	Setting	BBT2	RLPrompt	SoftP	P-tuning	Prefix	MetaL	MetaL (+Tune)
Large	cls→cls	22.43	23.13	28.53	34.01	39.71	35.62	**44.63**
	HR→LR	24.90	-	26.62	30.56	34.33	30.12	**35.46**
	QA→QA	24.92	-	20.66	27.14	26.52	28.94	**30.61**
	Avg.	24.08	-	25.27	30.57	33.52	31.56	**36.9**
XL	cls→cls	24.92	23.02	27.24	30.25	36.76	36.85	**46.00**
	HR→LR	25.31	-	24.33	29.36	34.15	30.05	**34.75**
	QA→QA	25.28	-	20.74	26.64	25.70	31.81	**34.03**
	Avg.	25.17	-	24.10	28.75	32.20	32.90	**38.26**

Table 4.4 Comparison of evaluation results on the prompt-only setting between MetaL-Prompt and the baselines. All examples are used for prompt generation or tuning, and no additional demonstration is provided for in-context learning.

4.4 Experimental Results

We evaluate MetaL-Prompt on the setup presented in Section 4.3. We first show the performance of MetaL-Prompt when the generated or tuned prompt is solely provided without additional demonstrations (Section 4.4.1). Then, we explore a trade-off between the number of examples used for in-context learning demonstrations for LMs and those for prompt generation (Section 4.4.2). We additionally validate whether the PGMs are capable of generalizing to inference settings that involve a different number of demonstrations from training settings (Section 4.4.3). Finally, we compare the performances of prompt designs we presented in Section 4.2.3 (Section 4.4.4).

4.4.1 Prompt-only In-context Learning

We present the performance of the prompts generated by MetaL-Prompt without extra demonstrations for in-context learning. In this setting, all 16 examples from a task are used for prompt generation or tuning. The inputs are composed of only the prompt and test inputs without any additional demonstrations ($n_P = 16, n_L = 0$). This setting is the most practical setting because it keeps the shortest input length. Short sequence length leads to low latency, small hidden state cache (i.e., key and value cache of transformer-based models for demonstrations or prompts), or low monetary cost for users.

As demonstrated in Table 4.4, MetaL-Prompt exhibits superior performance compared to the baselines, with the exception of Prefix-tuning. It achieves notable improvements of up to 19.4% on QA tasks, surpassing the state-of-the-art method (i.e., P-tuning), even with significantly lower computational costs requiring only a single forward pass. Interestingly, Prefix-tuning showcases exceptional efficacy in the limited budget setting compared to other baselines.

However, MetaL-Prompt outperforms Prefix-tuning on QA tasks, and maintains comparable performance on GPT2-XL, the larger model, and classification tasks. Notably, MetaL-Prompt benefits from its advantageously low computation costs while still delivering competitive results.

Our hypothesis regarding the lack of performance improvements in the HR→LR setting for MetaL-Prompt is attributed to the limited diversity of datasets in high-resource datasets. The PGMs employed by MetaL-Prompt leverage meta-knowledge obtained from a broad spectrum of tasks, emphasizing the significance of dataset diversity. However, it is worth noting that the meta-learning dataset specific to the HR→LR setting primarily comprises high-resource datasets. These datasets consist of tasks where corresponding data can be relatively easily collected. Consequently, we believe that the meta-learning dataset for this particular setting lacks the necessary variety of datasets required for effective meta-learning and prompt generation.

Furthermore, as discussed in Section 4.2.1, we can enable MetaL-Prompt to utilize comparable computation costs to the baselines (Section 4.3) by incorporating additional tuning of the generated prompts. The performances of this variant, referred to as *MetaL(+Tune)*, is presented in Table 4.4. When considering the fair computation costs, MetaL-Prompt consistently achieves superior performance across various task settings and models, and shows improvements up to 27.7% on the QA tasks compared to the state-of-the-art method, P-tuning.

Lastly, it is important to highlight that the gradient-based prompt tuning methods — SoftPrompt, P-tuning, and Prefix-tuning — demonstrate superior overall performance when applied to the smaller model particularly on the classification tasks. We attribute this phenomenon to the convergence speed. Due to the smaller size of the model, the prompts tuned by these methods converge more rapidly on the smaller model, allowing them to reach better performance

Method	Example Splits (16, 0)	(12, 4)
	cls→cls	
P-tuning	30.25	29.87
Prefix-tuning	36.76	37.07
MetaL-Prompt	36.85	39.33
	QA→QA	
P-tuning	26.64	26.53
Prefix-tuning	25.70	26.17
MetaL-Prompt	31.81	30.49

Table 4.5 Comparison of the performances on various distribution of examples for prompting and demonstration. We use GPT2-XL for the LM and PGMs.

within the constrained computational budgets.

However, it is noteworthy that MetaL-Prompt showcases better scalability across different model sizes compared to the baselines. This means that MetaL-Prompt is expected to deliver improved performance on larger models, whereas other gradient-based methods may encounter limitations when applied to such models. This scalability advantage positions MetaL-Prompt as a favorable choice for scenarios involving larger models.

4.4.2 Additional demonstrations with generated prompts

Although prompt-only is the most cost-efficient setting, model providers or users may be willing to allocate expanded computation budgets for inference or larger spatial budgets (i.e., larger key/value cache) to further increase the performance. For such situations, we explore the effect of additional demonstra-

tions combined with the generated prompt. We keep the number of examples for each task the same but vary the ratio between n_P and n_L. We evaluate MetaL-Prompt and two baselines, P-tuning [42] and Prefix-tuning [39], which are the most powerful baselines in Section 4.4.1, on settings where (n_P, n_L) is $(16, 0)$ and $(12, 4)$.

Table 4.5 presents the results of MetaL-Prompt and the baselines when additional demonstrations are provided. Notably, when given these extra demonstrations, MetaL-Prompt demonstrates further performance improvements, particularly in the cls→cls setting, with enhancements of up to 6.7%. Hence, depending on tasks, a model provider may opt to allocate a larger computation budget for inference on longer sequences or allocate additional spatial resources to cache the hidden states of the demonstrations, thereby enhancing performance. It is worth mentioning that Prefix-tuning also benefits from these budgetary allocations in both settings, whereas P-tuning does not exhibit the same advantage.

4.4.3 Transferability to different test settings

In the previous section, we have discussed that MetaL-Prompt can further enhance performance by tailoring example splits through increased computation or spatial budgets. In this section, we further explore that a PGM trained with a specific training setting (i.e., a particular example split) is still available in different test settings. We evaluate two PGMs trained for GPT2-XL where (n_P, n_L) is $(16, 0)$, $(12, 4)$. These models represent training without demonstrations and with additional demonstrations respectively. We evaluate the models on cls→cls and QA→QA with three test settings where (n_P, n_L) is $(16, 0)$, $(12, 4)$, and $(8, 8)$.

As presented in Table 4.6, PGMs trained without demonstrations (i.e., $(n_P, n_L) = (16, 0)$) does not generalize to the other test settings. The per-

Train Setting	Test Setting (16, 0)	(12, 4)	(8, 8)
	cls→cls		
(16, 0)	36.85	31.09	27.75
(12, 4)	32.46	39.33	39.38
	QA→QA		
(16, 0)	31.81	26.51	27.35
(12, 4)	23.56	30.49	31.31

Table 4.6 Results for transferability of the PGMs. We evaluate the PGMs for GPT2-XL on the test settings different from the training settings.

formance decreases when examples are provided with an LM as demonstrations instead of solely utilized for prompt generation. Interestingly, PGMs trained with $(n_P, n_L) = (12, 4)$, where demonstrations are considered, show marginal or no degradation when transferred to the other test settings with demonstrations.

In summary, PGMs trained without demonstrations can not be transferred to test settings with demonstrations, and vice versa. From this observation, we notice that there exists a significant disparity between suitable prompts that are soley used without demonstrations and those that are paired with demonstrations. Exploration of this gap will be an interesting future work.

4.4.4 Comparison between various prompt designs

In this section, we compare performances of diverse prompt designs listed in Section 4.2.3: *Discrete*, *Weighted Sum (WS)*, *Hidden State (HS)*, and *Prefix*. Discrete is a prompt consisting of natural language tokens, whereas Weighted

Method	Disc	WS	HS	Prefix
F1	31.78	33.13	31.19	36.85

Table 4.7 Ablation studies on the effect of our prompt design. We evaluate various prompt designs on GPT2-XL and cls→cls.

Sum, Hidden State, and Prefix represent continuous prompts, which are real-valued prompts. We compare the designs on cls→cls where (n_P, n_L) is $(16, 0)$.

As depicted in Table 4.7, Prefix exhibits the best performance among the approaches due to its ability to prepend to each layer, resulting in a larger prompt size while maintaining the same prompt length. This larger prompt size provides Prefix with a richer expressiveness compared to the other methods. Weighted Sum also demonstrates improved performance, benefiting from its enhanced expressiveness compared to the discrete prompt, which consists of natural language tokens.

However, Hidden State displays degenerated performance, even when compared to the discrete prompts. As discussed in Section 4.2.3, the input hidden states of the head layer have representations similar to word embeddings, but they are not identical, particularly in terms of the scale of the values. This discrepancy may cause the Hidden State prompts to deviate from the manifold of the word embeddings. We hypothesize that the degraded performance of Hidden State prompts is a result of this discrepancy. Consequently, incorporating an additional scaler to mitigate the discrepancy is expected to be beneficial in improving the performance of Hidden State prompts.

4.5 Discussion

In this section, we aim to highlight the limitations of MetaL-Prompt, and propose potential avenues for future research.

Input templates for prompt generation

MetaL-Prompt simply prepends generated prompts, which intuitively correspond to task instructions, to queries, but hand-crafted prompts are usually more complex. Hand-crafted prompts may include some tags for each data field or special characters to split demonstration examples, which are templates to be filled with data instances. However, MetaL-Prompt does not generate such input templates during prompt generation. Moreover, previous works have not yet systematically explored how the templates affect the performances if given in prompt generation. We will more deeply explore the generation of input templates, or the effects of input templates in prompt generation.

Flexibility on the number of demonstrations

As highlighted in Section 4.4.3, it is important to note that a prompt generation model (PGM) trained on a specific training setting (i.e., example split) cannot be directly transferred to different test settings. Consequently, when we aim to enhance prediction quality through the inclusion of additional demonstrations as on the cls→cls setting in Section 4.4.3, we require multiple PGMs for each specific setting, allowing for a flexible trade-off between inference speed and prediction quality by adjusting the number of demonstrations. However, training and managing multiple PGMs pose challenges for LMaaS providers.

Prompt generation with more examples

In our experiments, the prompt generation models (PGMs) are constrained by sequence size limits. Consequently, if a user provides an excessive number of examples, the PGMs may be unable to process such a large set if the concatenation of the examples exceeds the sequence size limit.

However, it is worth noting that recent language models have started to adopt longer sequence sizes, which helps alleviate this limitation. The incorporation of longer sequence sizes enables PGMs to handle larger sets of examples more effectively.

Additionally, we explore an iterative approach to improving prompts by concatenating a previously generated prompt with new examples. This concatenated context is then used as input to the PGM to generate an enhanced prompt, allowing the PGM to accommodate an arbitrary number of examples by iteratively processing the example subsets. This iterative approach enhances the scalability of prompt generation, empowering PGMs to process varying numbers of examples effectively.

Applying CoRe to MetaL-Prompt

As outlined in Figure 4.2 and Equation 4.1, MetaL-Prompt incorporates prompt gradients in the backpropagation process to compute gradients of the PGM parameters. Hence, intuitively, CoRe aids in generalization of PGM by regularizing the prompt gradients. If we apply CoRe to meta-learning of MetaL-Prompt, Equation 4.1 changes as follows:

$$\begin{aligned} \boldsymbol{\theta}_P^* &= \arg\max_{\boldsymbol{\theta}_P} \sum_i \sum_j p(y_{i,j} | \boldsymbol{\omega}, x_i^0, y_i^0, \boldsymbol{\omega}, x_i^1, y_i^1, ..., \boldsymbol{\omega}, x_i^j; \boldsymbol{\theta}_L) \\ s.t. \quad & \boldsymbol{\omega} = f_{\boldsymbol{\theta}_P}(X_i^P), \end{aligned} \tag{4.2}$$

where j is the sequence size of CoRe. Although the effectiveness of CoRe on MetaL-Prompt has not been validated yet, it is a promising area for further exploration.

Chapter 5

Conclusion

5.1 Conclusion

In this dissertation, we explored leveraging contexts induced by input sequences in language models (LMs) to propose novel automatic prompt engineering methods for in-context learning. As presented in Table 2.1, we first present CoRe to improve gradient-based prompt tuning methods, and we also introduce MetaL-Prompt which is a lightweight prompting method for LMaaS.

CoRe is a novel gradient-based prompt tuning method that regularizes task contexts among multiple examples, which finally regularizes a prompt to improve zero-shot performance. Two regularizers of CoRe— context attuning and context filtering— regularize the prompt to create proper task context and guides the prompt to convey only the task-related context to succeeding examples on the concatenation of multiple examples. We provide a theoretical analysis for the effect of context attuning and context filtering and our experimental results back this up. Following the theoretical analysis, CoRe achieves

performance gain over three different baseline prompt tuning methods in zero-shot setting up to 11.9% on GPT2-XL and 6.3% on GPT-J.

MetaL-Prompt is a novel lightweight prompting method for LMaaS based on meta-learning of prompt generation. MetaL-Prompt trains a prompt generation model (PGM), which predicts a prompt given examples for a task, using our meta-learning objective. After the meta-learning, the trained PGM does not require additional training for unseen user tasks. For efficient training of PGM, we also propose trainable padding, which approximates the generation process in meta-learning and mitigates the overhead. Moreover, we explore the prediction of continuous prompts using an LM. This has not yet been discussed in the previous prompting studies, which search for the best prompt by repeatedly manipulating a prompt. With the proposed designs, MetaL-Prompt achieves performance gains over five baselines up to 19.4% on unseen QA datasets with much less computation than the baselines. When given fair computation budgets, MetaL-Prompt shows improvements up to 27.7% on unseen QA datasets using additional tuning of the generated prompts. The results support the efficiency of MetaL-Prompt in terms of model performance and computational cost.

5.2 Future Work

5.2.1 Interpretation of in-context learning

Despite the success of in-context learning, there is no study that interprets task knowledge that an LM learns from a context. Some methods lead an LM to generate an explanation of a task from demonstration examples [24, 72] using hand-crafted instructions. However, the heuristic approach of the hand-crafted instructions does not guarantee that the LM actually explains how it

understands the examples.

MetaL-Prompt in chapter 4 demonstrated that a tuned LM can more robustly extract task information from examples and pack the information in some embeddings that the target LM can interpret compared to the prompt generation methods. Therefore, approaches of MetaL-Prompt can be used to interpret in-context learning. Since MetaL-Prompt now only support continuous prompts, which is not human-readable, we need to develop a training scheme for discrete token. Moreover, we may need an additional approach to make a prompt generation model generate real natural language texts considering that discrete prompts are still sometimes not human-readable [60].

5.2.2 Applicability of automatic prompt engineering according to tasks

As mentioned in the preceding sections, both CoRe and MetaL-Prompt do not consistently exhibit gains on certain tasks. This phenomenon raises questions about the specific task characteristics that hinder the effectiveness of automatic prompt engineering methods. The effectiveness of prompt engineering on diverse tasks remains inadequately explored. Therefore, it is crucial to assess the validity of other automatic prompt engineering methods and our context-based approaches across diverse tasks. By doing so, we can identify tasks where these methods may prove ineffective and subsequently investigate the task characteristics responsible for disabling automatic prompt engineering.

5.2.3 Location of adaptation modules

Many previous works on LM adaptation have proposed adaptation modules, including prompts, that is placed to various location of a model [29, 28, 45, 2, 39]. The modules are placed on prefixes of keys and values [39, 61], after feed-forward

module [28], in parallel to linear layers [29], or on other somewhere. The methods have shown remarkable performances but the effect of locations of adaptation modules has not been properly explored. He et al. [23] analyzed several locations proposed by previous works, but they explore limited locations on a small number of datasets. Therefore, more extended analyses are encouraged.

Bibliography

[1] A. Antoniou, H. Edwards, and A. Storkey. How to train your maml. In *International Conference on Learning Representations*, 2019.

[2] E. Ben Zaken, Y. Goldberg, and S. Ravfogel. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.1. URL `https://aclanthology.org/2022.acl-short.1`.

[3] O. Bohdal, Y. Yang, and T. Hospedales. Flexible dataset distillation: Learn labels instead of images. *arXiv preprint arXiv:2006.08572*, 2020.

[4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[5] Y. Chai, S. Wang, Y. Sun, H. Tian, H. Wu, and H. Wang. Clip-tuning: Towards derivative-free prompt learning with a mixture of rewards. In *Findings of the Association for Computational Linguistics: EMNLP 2022*,

pages 108–117, Abu Dhabi, United Arab Emirates, Dec. 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.findings-emnlp.8.

[6] W. Chen, H. Wang, J. Chen, Y. Zhang, H. Wang, S. Li, X. Zhou, and W. Y. Wang. Tabfact: A large-scale dataset for table-based fact verification. In *International Conference on Learning Representations*, 2020.

[7] Y. Chen, R. Zhong, S. Zha, G. Karypis, and H. He. Meta-learning via language model in-context tuning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 719–730, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.53. URL https://aclanthology.org/2022.acl-long.53.

[8] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[9] C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*, 2019.

[10] CLOVA. Clova studio, 2022. https://www.ncloud.com/product/aiService/clovaStudio.

[11] J. Davison, J. Feldman, and A. M. Rush. Commonsense knowledge mining from pretrained models. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 1173–1178, 2019.

[12] M.-C. de Marneffe, M. Simons, and J. Tonhauser. The commitmentbank: Investigating projection in naturally occurring discourse. 2019.

[13] M. Deng, J. Wang, C.-P. Hsieh, Y. Wang, H. Guo, T. Shu, M. Song, E. P. Xing, and Z. Hu. Rlprompt: Optimizing discrete text prompts with reinforcement learning. *arXiv preprint arXiv:2205.12548*, 2022.

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://aclanthology.org/N19-1423`.

[15] S. Diao, Z. Huang, R. Xu, X. Li, Y. Lin, X. Zhou, and T. Zhang. Black-box prompt learning for pre-trained language models. *Transactions on Machine Learning Research*, 2023.

[16] N. Ding, Y. Qin, G. Yang, F. Wei, Z. Yang, Y. Su, S. Hu, Y. Chen, C.-M. Chan, W. Chen, et al. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*, 2022.

[17] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, 2017.

[18] T. Gao, A. Fisch, and D. Chen. Making pre-trained language models better few-shot learners. In *Proceedings of the 59th Annual Meet-*

ing of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 3816–3830, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.295. URL `https://aclanthology.org/2021.acl-long.295`.

[19] GooseAI. Gooseai, 2021. `https://goose.ai/`.

[20] A. Gulli. Ag's corpus of news articles. URL `http://groups.di.unipi.it/~gulli/AG_corpus_of_news_articles.html`.

[21] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001. doi: 10.1162/106365601750190398.

[22] N. Hansen, S. D. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary Computation*, 11(1):1–18, 2003. doi: 10.1162/106365603321828970.

[23] J. He, C. Zhou, X. Ma, T. Berg-Kirkpatrick, and G. Neubig. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*, 2022.

[24] O. Honovich, U. Shaham, S. R. Bowman, and O. Levy. Instruction induction: From few examples to natural language task descriptions. *arXiv preprint arXiv:2205.10782*, 2022.

[25] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169, 2021.

[26] B. Hou, J. O'Connor, J. Andreas, S. Chang, and Y. Zhang. Promptboosting: Black-box text classification with ten forward passes. *arXiv preprint arXiv:2212.09257*, 2022.

[27] Y. Hou, H. Dong, X. Wang, B. Li, and W. Che. Metaprompting: Learning to learn better prompts. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 3251–3262, 2022.

[28] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[29] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=nZeVKeeFYf9`.

[30] HuggingFace. Distilgpt2, 2019. `https://huggingface.co/distilgpt2`.

[31] Z. Jiang, A. Anastasopoulos, J. Araki, H. Ding, and G. Neubig. X-factr: Multilingual factual knowledge retrieval from pretrained language models. *arXiv preprint arXiv:2010.06189*, 2020.

[32] Z. Jiang, J. Araki, H. Ding, and G. Neubig. How can we know when language models know? on the calibration of language models for question answering. *Transactions of the Association for Computational Linguistics*, 9:962–977, 2021. doi: 10.1162/tacl_a_00407. URL `https://aclanthology.org/2021.tacl-1.57`.

[33] D. Khashabi, S. Chaturvedi, M. Roth, S. Upadhyay, and D. Roth. Looking

beyond the surface:a challenge set for reading comprehension over multiple sentences. In *Proceedings of North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018.

[34] D. Khashabi, S. Min, T. Khot, A. Sabharwal, O. Tafjord, P. Clark, and H. Hajishirzi. Unifiedqa: Crossing format boundaries with a single qa system. *arXiv preprint arXiv:2005.00700*, 2020.

[35] B. Lester, R. Al-Rfou, and N. Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.243. URL `https://aclanthology.org/2021.emnlp-main.243`.

[36] B. Lester, R. Al-Rfou, and N. Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, 2021.

[37] Y. Levine, N. Wies, D. Jannai, D. Navon, Y. Hoshen, and A. Shashua. The inductive bias of in-context learning: Rethinking pretraining example design, 2021. URL `https://arxiv.org/abs/2110.04541`.

[38] Q. Lhoest, A. Villanova del Moral, Y. Jernite, A. Thakur, P. von Platen, S. Patil, J. Chaumond, M. Drame, J. Plu, L. Tunstall, J. Davison, M. Šaško, G. Chhablani, B. Malik, S. Brandeis, T. Le Scao, V. Sanh, C. Xu, N. Patry, A. McMillan-Major, P. Schmid, S. Gugger, C. Delangue, T. Matussière, L. Debut, S. Bekman, P. Cistac, T. Goehringer, V. Mustar, F. Lagunas, A. Rush, and T. Wolf. Datasets: A community library for natural language processing. In *Proceedings of the 2021*

Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 175–184, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics. URL https://aclanthology.org/2021.emnlp-demo.21.

[39] X. L. Li and P. Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353.

[40] J. Liu, Y. Bai, G. Jiang, T. Chen, and H. Wang. Understanding why neural networks generalize well through gsnr of parameters. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HyevIJStwH.

[41] J. Liu, D. Shen, Y. Zhang, B. Dolan, L. Carin, and W. Chen. What makes good in-context examples for GPT-3? In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 100–114, Dublin, Ireland and Online, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.deelio-1.10. URL https://aclanthology.org/2022.deelio-1.10.

[42] X. Liu, Y. Zheng, Z. Du, M. Ding, Y. Qian, Z. Yang, and J. Tang. Gpt understands, too. *arXiv:2103.10385*, 2021.

[43] X. Liu, K. Ji, Y. Fu, W. Tam, Z. Du, Z. Yang, and J. Tang. P-tuning:

Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 61–68, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.8. URL `https://aclanthology.org/2022.acl-short.8`.

[44] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[45] Y. Mao, L. Mathias, R. Hou, A. Almahairi, H. Ma, J. Han, S. Yih, and M. Khabsa. Unipelt: A unified framework for parameter-efficient language model tuning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6253–6264, 2022.

[46] Microsoft. Introduction to prompt engineering, 2023. `https://learn.microsoft.com/en-us/azure/cognitive-services/openai/concepts/prompt-engineering`.

[47] S. Min, M. Lewis, L. Zettlemoyer, and H. Hajishirzi. Metaicl: Learning to learn in context. *arXiv preprint arXiv:2110.15943*, 2021.

[48] A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[49] OpenAI. Openai api, 2020. `https://openai.com/api/`.

[50] F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, and S. Riedel. Language models as knowledge bases? *arXiv preprint arXiv:1909.01066*, 2019.

[51] M. T. Pilehvar and J. Camacho-Collados. WiC: the word-in-context dataset for evaluating context-sensitive meaning representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1267–1273, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1128. URL `https://aclanthology.org/N19-1128`.

[52] A. Prasad, P. Hase, X. Zhou, and M. Bansal. Grips: Gradient-free, edit-based instruction search for prompting large language models, 2022. URL `https://arxiv.org/abs/2203.07281`.

[53] R. Puri and B. Catanzaro. Zero-shot text classification with generative language models. *arXiv preprint arXiv:1912.10165*, 2019.

[54] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.

[55] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.

[56] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL `https://arxiv.org/abs/1908.10084`.

[57] M. Roemmele, C. Bejan, and A. Gordon. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. 01 2011.

[58] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. In *NeurIPS EMC^2 Workshop*, 2019.

[59] T. Schick and H. Schütze. Exploiting cloze questions for few shot text classification and natural language inference. *arXiv preprint arXiv:2001.07676*, 2020.

[60] T. Shin, Y. Razeghi, R. L. L. IV, E. Wallace, and S. Singh. AutoPrompt: Eliciting knowledge from language models with automatically generated prompts. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[61] T. Sun, Z. He, H. Qian, Y. Zhou, X. Huang, and X. Qiu. Bbtv2: Towards a gradient-free future with large language models. In *Proceedings of EMNLP*, 2022.

[62] T. Sun, Y. Shao, H. Qian, X. Huang, and X. Qiu. Black-box tuning for language-model-as-a-service. In *International Conference on Machine Learning*, pages 20841–20855. PMLR, 2022.

[63] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019.

[64] T. Wang, J.-Y. Zhu, A. Torralba, and A. A. Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

[65] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen,

C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2020.emnlp-demos.6`.

[66] S. M. Xie, A. Raghunathan, P. Liang, and T. Ma. An explanation of in-context learning as implicit bayesian inference. In *International Conference on Learning Representations*, 2022.

[67] Q. Ye, B. Y. Lin, and X. Ren. Crossfit: A few-shot learning challenge for cross-task generalization in nlp. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7163–7189, 2021.

[68] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-28-1. URL `https://www.usenix.org/conference/osdi22/presentation/yu`.

[69] T. Zhang, X. Wang, D. Zhou, D. Schuurmans, and J. E. Gonzalez. Tempera: Test-time prompt editing via reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2023.

[70] X. Zhang, J. Zhao, and Y. LeCun. Character-level Convolutional Networks for Text Classification. *arXiv:1509.01626 [cs]*, Sept. 2015.

[71] Z. Zhong, D. Friedman, and D. Chen. Factual probing is [mask]: Learning vs. learning to recall. In *North American Association for Computational Linguistics (NAACL)*, 2021.

[72] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba. Large language models are human-level prompt engineers. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

초록

프롬프팅은 대규모 언어 모델(LMs)의 적응에 효과적인 방법으로 큰 관심을 받고 있다. 프롬프트는 일반적으로 과제에 대한 설명과 예제를 포함하고, 이는 LM의 입력으로 제공된다. LM은 프롬프트를 통해서 주어진 맥락을 통해 과제를 이해하고 실제로 풀어야 하는 문제를 처리하고, 이 방식을 인-컨텍스트 학습이라고 한다. 하지만 프롬프트는 종종 인간의 직관에 반하여 불안정한 성능을 보여주기 때문에 기존 연구들은 효과적인 프롬프트를 자동으로 찾는 방법들을 제안하게 된다.

자동 프롬프트 엔지니어링 방법은 다양한 NLP 작업에서 뛰어난 성능을 보여주지만, 특정 시나리오에서는 전체 파라미터 파인튜닝과 같은 몇 가지 LM 적응 방법에 비해 성능이 떨어진다. 최적의 성능을 보여주지 못하더라도, 간단한 디자인이나 파라미터 효율성 등 프롬프트의 특수한 장점은 프롬프트의 성능을 개선하기 위한 방법을 연구의 동기가 된다.

또한, 프롬프트 튜닝 방법은 LM 적응에 효과적이지만, 언어 모델 서비스 (language-model-as-a-service (LMaaS)를 지원하기 위해 설계되지 않았다. 최근 대규모 언어 모델은 주로 서비스 형태(LMaaS)로 제공된다. LMaaS는 모델의 파라미터를 공개하지 않기 때문에, 사용자는 서비스를 사용할 때 인-컨텍스트 학습을 위해 과제별 프롬프트를 준비해야 한다. 하지만, LMaaS는 무거운 계산 비용 때문에 자동 프롬프트 튜닝 방법을 서비스 내에서 제공하지 않는다. LMaaS 사용자는 파라미터 접근이 필요하지 않거나 서비스 제공자의 추가 지원이 필요하지 않는 여러 블랙박스 프롬프트 방법을 사용할 수 있지만, 비전문가 사용자가 본인의 장비에서 이러한 방법을 배포하고 실행하기는 매우 어렵다.

이 논문에서는 먼저 CoRe라는 새로운 정규화 방법을 제안한다. 이 방법은 gradient 기반 프롬프트 튜닝 기술에 적용되어 프롬프트가 과제에 대한 맥락을 올바르게 생성하도록 유도한다. CoRe는 컨텍스트 어튜닝과 컨텍스트 필터링이라는

두 가지 정규화 효과를 실현하여, 과제에 대한 예시 없이 CoRe에 의해 튜닝된 프롬프트만을 사용하여 인퍼런스가 이루어지는 "제로-샷 인-컨텍스트 러닝" 환경에서 예측 성능을 향상시킨다. 컨텍스트 어튜닝은 입력과 튜닝된 프롬프트에 의해 생성된 맥락이 작업에 적합한 맥락을 담도록 유도한다. 이론적 분석을 통해 맥락의 정규화는 제로-샷 인-컨텍스트 러닝 성능을 향상시키는 데 기여한다는 것을 알 수 있다. 컨텍스트 필터링은 프롬프트가 작업과 관련된 맥락에 집중하도록 유도하여 컨텍스트 어튜닝이 올바른 작업 맥락를 생성하고 전송하는 데에만 집중하도록 합니다. 우리는 자연어 이해 데이터셋과 GPT2-XL 및 GPT-J라는 두 가지 대규모 언어 모델에서 CoRe를 평가한다. CoRe는 제로-샷 설정에서 GPT2-XL에서 최대 11.9%의 성능 향상과 GPT-J에서 최대 6.3%의 성능 향상을 보여준다.

그리고 우리는 MetaL-Prompt라는 LMaaS를 위한 새로운 경량 프롬프트 생성 방법을 제안한다. MetaL-Prompt는 적은 수의 데이터를 활용하여 추가적인 훈련 없이 해당 작업에 대한 프롬프트를 생성하는 프롬프트 생성 모델(PGM)을 메타러닝을 통해 학습한다. 또한, 메타러닝 도중 또는 프롬프트 생성 도중의 생성 과정으로 인한 부하를 완화하기 위해 trainable padding을 제안하고, 프롬프트 생성 모델을 사용하여 다양한 프롬프트 유형의 생성을 탐구합니다. MetaL-Prompt는 PGM이 특정 과제에 대한 예제들의 연결로 인해 발생하는 컨텍스트에서 과제에 대한 정보를 추출하고, 이를 기반으로 단일 포워드 패스를 통해 프롬프트를 생성하기 때문에 계산 측면에서 효율적이다. 따라서, MetaL-Prompt는 LMaaS에 적용되었을 때 계산 부하가 적으며, 서비스는 자동으로 생성된 프롬프트를 사용하여 다양한 작업을 지원할 수 있다. 우리는 다양한 메타러닝 설정에서 MetaL-Prompt를 평가하였으며, 제로-샷 인-컨텍스트 러닝 환경에서 최신 베이스라인과 비교하여 QA 데이터셋에서 평균 F1 점수를 최대 19.4%까지 향상시킨다. 또한 이를 달성하는데 베이스라인에 비해 아주 적은 계산 비용이 든다.

주요어: 프롬프트 튜닝, 프롬프트, 인-컨텍스트 학습, 언어 모델
학번: 2016-21234