nature machine intelligence

Analysis

https://doi.org/10.1038/s42256-023-00626-4

Parameter-efficient fine-tuning of large-scale pre-trained language models

Received: 13 April 2022

Accepted: 2 February 2023

Published online: 2 March 2023

Check for updates

Ning Ding[1,2,4], Yujia Qin[1,2,4], Guang Yang[1], Fuchao Wei[1], Zonghan Yang[1], Yusheng Su[1,2], Shengding Hu[1,2], Yulin Chen[3], Chi-Min Chan[1], Weize Chen[1,2], Jing Yi[1,2], Weilin Zhao[1,2], Xiaozhi Wang[1], Zhiyuan Liu[1,2] ✉, Hai-Tao Zheng[3] ✉, Jianfei Chen[1], Yang Liu[1], Jie Tang[1,2], Juanzi Li[1] & Maosong Sun[1,2] ✉

With the prevalence of pre-trained language models (PLMs) and the pre-training–fine-tuning paradigm, it has been continuously shown that larger models tend to yield better performance. However, as PLMs scale up, fine-tuning and storing all the parameters is prohibitively costly and eventually becomes practically infeasible. This necessitates a new branch of research focusing on the parameter-efficient adaptation of PLMs, which optimizes a small portion of the model parameters while keeping the rest fixed, drastically cutting down computation and storage costs. In general, it demonstrates that large-scale models could be effectively stimulated by the optimization of a few parameters. Despite the various designs, here we discuss and analyse the approaches under a more consistent and accessible term 'delta-tuning', where 'delta' a mathematical notation often used to denote changes, is borrowed to refer to the portion of parameters that are 'changed' during training. We formally describe the problem and propose a unified categorization criterion for existing delta-tuning methods to explore their correlations and differences. We also discuss the theoretical principles underlying the effectiveness of delta-tuning and interpret them from the perspectives of optimization and optimal control. Furthermore, we provide a holistic empirical study on over 100 natural language processing tasks and investigate various aspects of delta-tuning. With comprehensive study and analysis, our research demonstrates the theoretical and practical properties of delta-tuning in the adaptation of PLMs.

With the revolutionary development in computing hardware, traditional statistical methods for modelling natural language have yielded their place to deep learning[1] that heavily relies on tensor computation and huge data volume. Modern natural language processing (NLP) uses deep neural networks to implicitly model language distribution and capture language representations[2–4]. A standard pipeline involves encoding language into discrete tokens (tokenization) as model input, choosing a proper model architecture, designing corresponding tasks and training the network with the given corpora. Among these deep neural architectures, the transformer neural network[4] produces state-of-the-art performances on a series of NLP applications. Subsequently, the advancement in pre-trained language models (PLMs) using deep transformers as their foundation has ushered in a new era of NLP. PLMs typically use heavily over-parameterized transformers as the base architecture and model natural language in bidirectional[5], autoregressive[6,7] or sequence-to-sequence[8] manners on large-scale

[1]Department of Computer Science and Technology, Tsinghua University, Beijing, China. [2]Beijing Academy of Artificial Intelligence, Beijing, China. [3]Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen, China. [4]These authors contributed equally: Ning Ding, Yujia Qin. ✉e-mail: liuzy@tsinghua.edu.cn; zheng.haitao@sz.tsinghua.edu.cn; sms@tsinghua.edu.cn

unsupervised corpora. Then for downstream tasks, task-specific objectives are introduced to fine-tune the PLMs for model adaptation. Notably, the increasing scale of PLMs (measured by the number of parameters) seems to be an irreversible trend, as constant empirical results show that larger models (along with more data) almost certainly lead to better performance. For example, with 175 billion parameters, Generative Pre-trained Transformer 3 (GPT-3)[9] generates natural language of unprecedented quality and can conduct various desired zero-shot tasks with satisfactory results given appropriate prompts. Subsequently, a series of large-scale models such as Gopher[10], Megatron-Turing Natural Language Generation (NLG)[11] and Pathways Language Model (PaLM)[12] have repeatedly shown effectiveness on a broad range of downstream tasks.

As the model scales, how to efficiently and effectively adapt large models to particular downstream tasks becomes an intriguing research issue. Although in-context learning has shown promising performance for PLMs such as GPT-3, fine-tuning still overtakes it under the task-specific setting. However, the predominant approach, full parameter fine-tuning, which initializes the model with the pre-trained weights, updates all the parameters and produces separate instances for different tasks, becomes impractical when dealing with large-scale models. In addition to the cost of deployment and computation, storing different instances for different tasks is extremely memory intensive. To further explore the practical application rate of large models (PLMs with over 1 billion parameters), we randomly select 1,200 published research papers from the recent six NLP conferences (200 for each venue), including Annual Meeting of the Association for Computational Linguistics (ACL) 2022, ACL 2021, Conference on Empirical Methods in Natural Language Processing (EMNLP) 2021, Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL) 2021, ACL 2020 and EMNLP 2020. Then we manually count the usage of PLMs in these peer-reviewed works, focusing on only the experimental part of the papers. According to the statistics in Extended Data Table 1, although the use of PLMs has become increasingly popular, only about 0.5–4% of research papers practically adopt large PLMs in the experiments. One of the reasons for their unpopularity is the unaffordable cost of deploying and experimentally validating large PLMs.

In fact, large PLMs with billions of parameters could be effectively driven by optimization of a few parameters, and a branch of parameter-efficient methods for model tuning arises. Although each of these approaches proposes distinct designs on the structure and location of trainable parameters in PLMs, they essentially tune a 'delta' in the adaptation phase, which refers to a small fraction of trainable parameters that can be placed anywhere in the PLM. We thus unify them under a more accessible term 'delta-tuning' that captures the essence of this branch of methods more precisely. In general, delta-tuning updates only a small number of parameters (inherently in the model or additionally introduced) while freezing the remaining parameters that account for the vast majority. Adapter tuning[13] is among the earliest approaches to steer pre-trained models with a limited number of parameters. It inserts adapter modules with bottleneck architecture between layers in PLMs and only these inserted modules get updated during fine-tuning. BitFit[14] updates the bias terms in PLMs while freezing the remaining modules. Low rank adaptation (LoRA)[15] decomposes attention weight update into low-rank matrices to reduce the number of trainable parameters. The delta-tuning methods enable efficient tuning and practical usage for large pre-trained models and often achieve comparable results to the standard fine-tuning. For example, the vanilla fine-tuning of GPT-3 needs to update about 175,255 million parameters, which is almost infeasible in both industry and academia. However, if we tune only the injected low-rank decomposition matrices in each transformer layer[15], only 37.7 million parameters will be involved in backpropagation. Delta-tuning not only provides a promising way to adapt large PLMs but also sheds light on the mechanisms behind such model adaptations. Compared with fine-tuning, delta-tuning makes model adaptation a considerably low-cost process. For instance, researchers find that the optimization problem of the adaptations for big models could be reparameterized into a low-dimensional 'intrinsic subspace'[16,17] and various NLP tasks could be handled by tuning only very few parameters in the subspace. The empirical evidence takes us one step closer to understanding how pre-trained models work and may even spawn new theoretical questions that are worth exploring.

This Analysis attempts to comprehensively analyse recent advances in delta-tuning to establish a deeper understanding of this branch of methods (Methods). We formally describe the problem and categorize delta-tuning methods into addition-based, specification-based and reparameterization-based methods as illustrated in Fig. 4, then we comprehensively introduce the technical details and empirical conclusions of each method. To better understand the inner connections among the delta-tuning methods and the mechanisms of model adaptation, we develop theoretical analyses of delta-tuning by proposing theoretical frameworks from two different perspectives: optimization and optimal control. Our theoretical discussion is summarized as follows.

1. Optimization. Based on the knowledge of a low intrinsic dimension in a large PLM, we show that delta-tuning is essentially a subspace-optimization method with respect to the solution space or functional space. The discussion justifies the designs of the existing delta-tuning methods and explains some phenomena in the experiments.
2. Optimal control. Inspired by the relationship between deep learning and optimal control theories, we interpret delta-tuning as seeking optimal controllers for PLMs. We propose an optimal control framework that unifies different delta-tuning approaches. Our analysis provides theoretical references for the novel design of delta-tuning methods.

In terms of empirical studies, we carry out extensive and systematic experiments (Results) on over 100 NLP tasks to rigorously explore the performances, combinability, the power of scale, transferability and so on. Our main findings are summarized as follows.

1. Performance. Delta-tuning yields consistent and non-trivial performance on more than 100 NLP tasks, showing that it is an effective and lightweight alternative to conventional fine-tuning. Among several representative delta-tuning methods, no single algorithm predominantly outperforms the others.
2. Convergence. Training stability is also one of our focuses. Although the convergence of delta-tuning is generally not as fast as that of full parameter fine-tuning, we find that it is more sensitive to the delta structures than the number of tunable parameters. Meanwhile, the larger the model is, the faster the training converges.
3. Efficiency. In terms of computational efficiency, which is the original motivation for the methods, delta-tuning could substantially improve computational and storage efficiency while achieving decent results, highlighting the promising practical value of adapting super-large PLMs.
4. Combinability. Combining multiple delta-tuning methods is more effective than a single method in most cases, despite that the optimal combination may vary for different PLM backbones, downstream tasks and data scales. This finding implies the existence of an optimal delta structure, and it is likely that such a structure cannot be obtained artificially, but could be generated automatically.
5. Power of scale. The power of scale (that is, both the performance and convergence are improved when the size of the PLM increases) is observed in all of the delta-tuning methods, even in unregulated neural modules. In other words, when the model size is large enough, only optimizing a random portion of parameters can achieve comparable performance to conventional fine-tuning.

https://doi.org/10.1038/s42256-023-00626-4

Table 1 | Overall (test) performance of over 100 NLP tasks comparing PT, PF, LR, AP and FT

Task	PT (BASE)	PT (LARGE)	PF	LR	AP	FT
Ratio of tunable parameters	0.03%	0.01%	7.93%	0.38%	2.38%	100%
Classification/sentiment analysis						
GLUE-SST2	92.20	94.95	92.66	94.04	93.35	**94.27**
ROTTEN_TOMATOES	88.36	91.84	**89.96**	89.30	89.20	89.77
FINANCIAL_PHRASEBANK	97.18	98.36	**98.36**	97.94	97.95	**98.36**
POEM_SENTIMENT	54.18	70.31	85.38	**86.80**	82.52	83.26
YELP_POLARITY	95.47	98.18	97.78	97.37	97.30	**97.92**
AVG. OF SENTIMENT ANALYSIS	85.48	90.73	92.83	**93.09**	92.06	92.72
Classification/emotion						
EMO	69.91	71.47	73.31	**76.13**	74.88	75.69
EMOTION	89.19	88.73	88.29	88.63	88.98	**89.25**
TWEET_EVAL-HATE	53.00	42.23	44.67	48.16	47.88	**51.33**
TWEET_EVAL-IRONY	58.02	69.73	76.00	76.75	73.88	**77.43**
TWEET_EVAL-OFFENSIVE	75.94	78.87	80.94	80.97	80.59	**82.05**
TWEET_EVAL-SENTIMENT	28.90	72.79	71.78	71.31	71.90	**71.98**
TWEET_EVAL-STANCE_ABORTION	32.59	61.42	61.47	**63.20**	62.61	61.72
TWEET_EVAL-STANCE_ATHEISM	56.28	67.58	71.54	71.77	71.27	**74.41**
TWEET_EVAL-STANCE_CLIMATE	47.61	52.43	52.86	55.92	**59.06**	57.38
TWEET_EVAL-STANCE_FEMINIST	29.65	51.63	56.27	57.41	**58.57**	58.51
TWEET_EVAL-STANCE_HILLARY	41.34	63.18	62.15	65.40	61.74	**66.41**
AVG. OF EMOTION	52.95	65.46	67.21	68.70	68.31	**69.65**
Classification/hate-speech detection						
ETHOS-DISABILITY	46.99	100.00	93.81	93.81	**100.00**	93.81
ETHOS-GENDER	63.84	77.08	77.44	**79.91**	**79.91**	74.48
ETHOS-NATIONAL_ORIGIN	44.30	81.77	81.77	**87.95**	84.72	84.72
ETHOS-RACE	84.36	97.06	94.54	**97.21**	94.27	**97.21**
ETHOS-RELIGION	93.02	93.02	96.35	93.02	96.35	**96.64**
ETHOS-DIRECTED_VS_GENERALIZED	76.86	86.64	94.76	92.29	**94.94**	**94.94**
HATE_SPEECH_OFFENSIVE	73.27	79.08	**75.22**	75.21	75.06	75.04
HATE_SPEECH18	75.57	74.45	79.42	79.59	80.86	**80.93**
HATEXPLAIN	50.98	67.62	66.06	68.03	**68.11**	68.02
AVG. OF HATE SPEECH DETECTION	67.69	84.08	84.37	85.22	**86.02**	85.09
Classification/natural language inference						
ANLI	25.85	44.96	43.88	45.27	49.19	**50.54**
GLUE-MNLI	35.43	86.12	82.21	83.74	83.90	**86.39**
GLUE-QNLI	52.34	93.01	87.48	92.02	91.58	**92.57**
GLUE-RTE	45.32	79.14	72.66	79.14	78.42	**80.58**
SCITAIL	91.02	95.47	93.04	93.80	94.04	**94.77**
SUPERGLUE-RTE	50.36	84.89	73.38	79.14	**82.01**	78.42
SICK	40.10	88.82	87.91	88.69	88.88	**89.15**
SUPERGLUE-CB	75.00	78.57	**100.00**	**100.00**	96.43	96.43
AVG. OF NATURAL LANGUAGE INFERENCE	51.93	81.37	80.07	82.73	83.06	**83.61**
Classification/fact checking						
CLIMATE_FEVER	15.47	33.42	38.03	39.35	37.48	**41.57**
LIAR	13.23	28.87	26.46	**28.67**	27.08	28.20
HEALTH_FACT	39.15	45.60	50.38	52.05	51.21	**54.19**
TAB_FACT	46.65	50.16	52.53	56.86	53.42	**57.34**
AVG. OF FACT CHECKING	28.63	39.51	41.85	44.23	42.30	**45.36**

https://doi.org/10.1038/s42256-023-00626-4

Table 1 (continued) | Overall (test) performance of over 100 NLP tasks comparing PT, PF, LR, AP and FT

Task	PT (BASE)	PT (LARGE)	PF	LR	AP	FT
Classification/paraphrase						
GLUE-QQP	84.65	86.21	84.62	86.87	85.93	**89.13**
MEDICAL_QUESTIONS_PAIRS	46.56	91.80	85.25	88.52	90.16	87.21
PAWS	49.60	91.27	92.07	93.39	92.91	**93.60**
GLUE-MRPC	67.65	88.24	87.25	87.25	87.25	**89.71**
AVG. OF PARAPHRASE	62.12	89.38	87.3	89.01	89.06	**89.91**
Classification/topic						
AG_NEWS	91.37	93.61	93.42	94.63	94.60	**95.19**
Classification/binary						
BOOLQ	61.28	77.43	77.55	80.00	78.47	**81.77**
MC_TACO	76.25	88.39	86.02	**88.13**	86.81	87.34
AVG. OF BINARY	68.77	82.91	81.79	84.07	82.64	**84.56**
Classification/other						
ADE_CORPUS_V2-CLASSIFICATION	41.76	94.42	93.25	**94.47**	93.91	94.27
DISCOVERY	0.18	18.83	16.67	18.98	18.41	**25.88**
GLUE-COLA	0.00	55.60	50.95	49.40	44.66	**51.53**
SMS_SPAM	95.80	97.46	97.14	97.14	**97.46**	97.11
SUPERGLUE-WIC	50.16	68.34	64.89	68.65	70.53	**71.79**
WIKI_QA	48.78	73.97	64.10	72.15	70.75	**74.41**
CIRCA	13.51	77.39	80.16	82.38	82.93	**84.69**
ONESTOP_ENGLISH	22.53	98.23	**100.00**	**100.00**	**100.00**	**100.00**
TREC	90.80	91.51	91.38	93.38	93.36	**94.81**
TREC-FINEGRAINED	80.63	88.18	90.04	**91.44**	90.00	91.27
AVG. OF OTHER CLASSIFICATION	44.42	76.39	74.86	76.80	76.2	**78.58**
Question answering/closed-book question answering						
FREEBASE_QA	1.90	6.71	2.63	3.75	5.86	**23.52**
LAMA-CONCEPTNET	15.25	26.12	22.63	34.96	43.62	**70.28**
LAMA-GOOGLE_RE	11.78	14.08	12.60	18.82	23.73	**24.88**
LAMA-SQUAD	3.23	16.13	**12.90**	9.68	3.23	9.68
LAMA-TREX	59.13	63.68	63.91	66.21	67.23	**69.12**
NUMER_SENSE	50.53	56.75	53.30	56.27	53.97	**57.32**
SEARCH_QA	7.14	19.17	8.70	10.17	9.72	**19.26**
WEB_QUESTIONS	11.90	19.58	15.87	18.78	20.63	**25.40**
HOTPOT_QA	65.95	76.41	73.76	76.13	74.65	**78.45**
AVG. OF CLOSED-BOOK QA	25.20	33.18	29.59	32.75	33.63	**41.99**
Question answering/multiple-choice question answering						
COSMOS_QA	7.30	10.98	9.91	10.78	10.85	**11.32**
DREAM	49.19	71.83	58.70	61.00	59.53	**62.42**
HELLASWAG	23.82	70.28	24.76	32.82	27.60	**41.90**
OPENBOOKQA	44.80	54.40	50.20	52.20	53.80	**57.00**
QASC	19.22	47.73	33.26	37.80	33.05	**43.63**
QUAREL	54.89	54.71	57.25	59.78	57.61	**62.50**
QUARTZ-NO_KNOWLEDGE	65.43	68.88	68.49	67.09	66.96	**69.39**
QUARTZ-WITH_KNOWLEDGE	64.03	85.97	71.56	74.23	73.72	**76.28**
RACE-HIGH	34.51	60.09	42.82	59.52	58.92	**65.95**
RACE-MIDDLE	47.21	74.65	62.67	68.31	65.46	**70.61**
SUPERGLUE-COPA	53.60	56.00	58.40	56.40	**60.40**	59.20
WINO_GRANDE	48.42	58.20	50.79	61.20	50.47	**67.19**
COMMONSENSE_QA	58.43	76.76	58.43	**62.52**	60.72	61.21

https://doi.org/10.1038/s42256-023-00626-4

Table 1 (continued) | Overall (test) performance of over 100 NLP tasks comparing PT, PF, LR, AP and FT

Task	PT (BASE)	PT (LARGE)	PF	LR	AP	FT
SCIQ	96.95	98.53	98.08	**98.42**	98.19	98.30
WIQA	36.10	65.27	63.67	77.99	64.44	**79.82**
AVG. OF MULTIPLE-CHOICE QA	46.93	63.62	53.93	58.67	56.11	**61.78**
Question answering/long-form question answering						
ELI5-ASKH	11.26	11.70	12.64	11.99	11.45	**13.00**
ELI5-ASKS	14.79	15.54	15.09	15.25	15.01	**15.28**
ELI5-ELI5	14.19	15.38	**15.23**	14.59	14.43	14.75
AVG. OF LONG-FORM QA	13.41	14.21	14.32	13.94	13.63	**14.34**
Question answering/machine reading comprehension						
SUPERGLUE-RECORD	44.67	73.82	61.62	64.66	62.08	**67.20**
MULTI_NEWS	18.09	19.23	18.81	19.44	19.10	**19.80**
ADVERSARIAL_QA	34.10	54.60	43.17	46.40	45.35	**48.56**
AVG. OF READING COMPREHENSION	32.29	49.22	41.20	43.50	42.18	**45.19**
Conditional generation/summarization						
SAMSUM	39.35	45.12	43.38	45.00	44.68	**45.73**
XSUM	21.35	26.56	23.84	25.87	26.07	**29.90**
AVG. OF SUMMARIZATION	30.35	35.84	33.61	35.44	35.38	**37.82**
Conditional generation/other						
SPIDER	3.29	6.38	7.74	**9.67**	8.70	6.77
WIKI_BIO	42.39	44.03	44.84	45.36	46.19	**47.09**
WIKI_SPLIT	79.80	80.10	79.91	80.09	80.05	**80.34**
AVG. OF OTHER GENERATION	41.83	43.50	44.16	**45.04**	44.98	44.73
Other/linguistic phenomenon						
BLIMP-ANAPHOR_GENDER_AGREEMENT	**100.00**	100.00	**100.00**	**100.00**	**100.00**	99.00
BLIMP-ELLIPSIS_N_BAR_1	49.00	100.00	**100.00**	**100.00**	**100.00**	**100.00**
BLIMP-SENTENTIAL_NEGATION	54.00	100.00	**100.00**	**100.00**	**100.00**	**100.00**
_NPI_SCOPE						
BLIMP-ANAPHOR_NUMBER_AGREEMENT	49.00	100.00	**100.00**	**100.00**	**100.00**	**100.00**
BLIMP-DETERMINER_NOUN_AGREEMENT	46.00	100.00	**100.00**	**100.00**	**100.00**	**100.00**
_WITH_ADJ_IRREGULAR_1						
BLIMP-EXISTENTIAL_THERE	53.00	100.00	**100.00**	**100.00**	**100.00**	**100.00**
_QUANTIFIERS_1						
BLIMP-IRREGULAR_PAST	**100.00**	100.00	**100.00**	**100.00**	**100.00**	**100.00**
_PARTICIPLE_ADJECTIVES						
BLIMP-WH_QUESTIONS_OBJECT_GAP	55.00	100.00	**100.00**	**100.00**	**100.00**	**100.00**
AVG. OF LINGUISTIC PHENOMENON	63.25	100.00	**100.00**	**100.00**	**100.00**	99.88
Other/generate explanation						
COS_E	12.41	14.82	13.90	14.05	**14.31**	13.46
Other/slot filling						
ADE_CORPUS_V2-DOSAGE	78.57	89.29	82.14	**85.71**	82.14	82.14
ADE_CORPUS_V2-EFFECT	59.15	61.35	**63.25**	62.52	60.91	62.66
AVG. OF SLOT FILLING	68.86	75.32	72.70	**74.12**	71.53	72.40
Other/other						
ACRONYM_IDENTIFICATION	93.35	96.68	**96.12**	**96.12**	95.57	**96.12**
ASLG_PC12	15.78	44.07	47.71	73.72	80.65	**92.92**
CRAWL_DOMAIN	68.16	76.91	73.04	73.00	72.76	**75.12**
PROTO_QA	21.16	37.66	24.57	27.87	26.17	**34.47**
AVG. OF OTHER TASKS	49.61	63.83	60.36	67.68	68.79	**74.66**
AVG. OF ALL TASKS	49.80	67.18	65.08	67.31	66.80	**69.27**

We experiment all methods on $T5_{BASE}$, with the best performance highlighted in bold, and also report the performance of PT on $T5_{LARGE}$.

https://doi.org/10.1038/s42256-023-00626-4

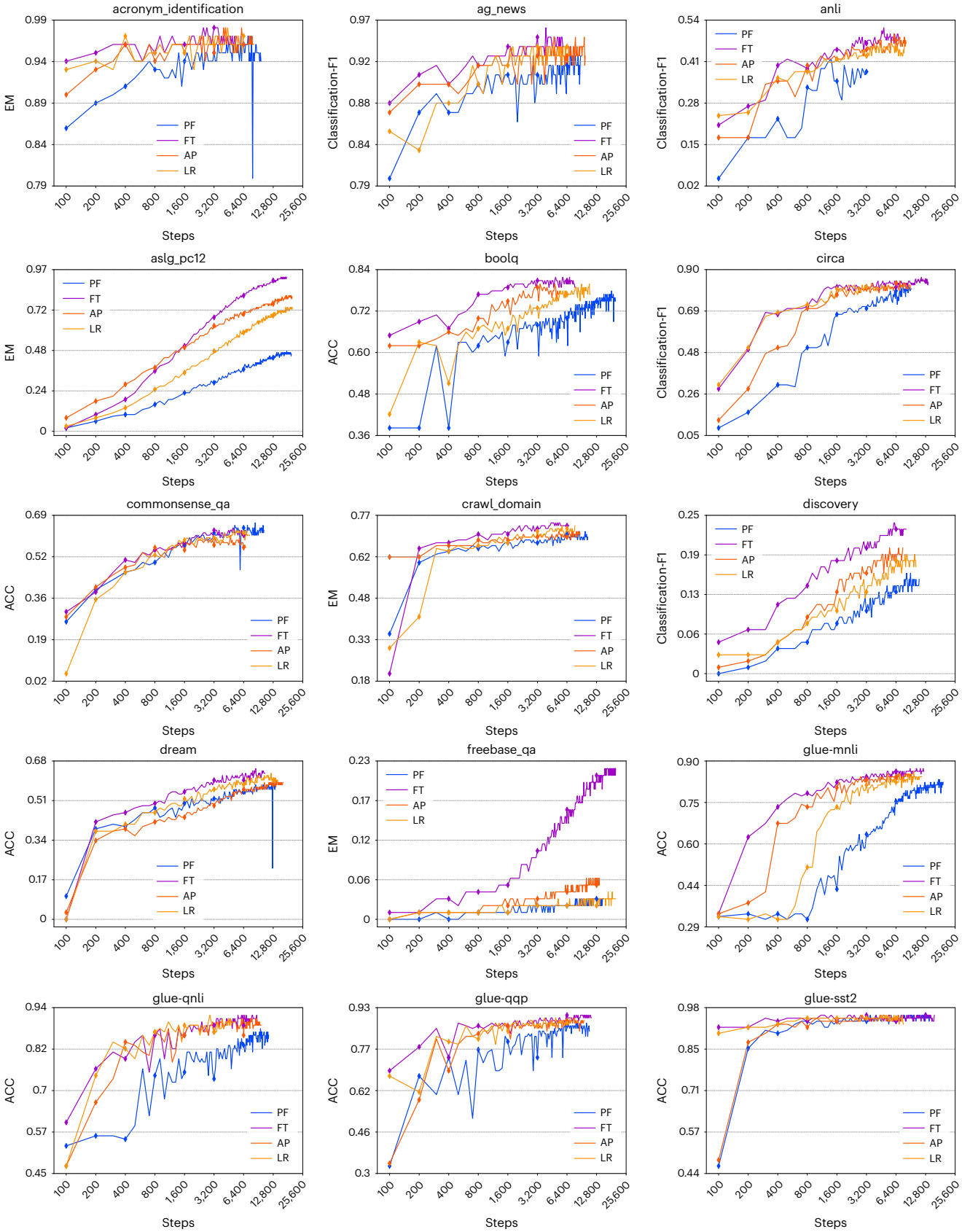


Fig. 1 | The performance of $T5_{BASE}$ with different delta-tuning methods (LR, AP and PF) and fine-tuning (FT) at different training steps. Note we apply early stopping to all methods. We choose three metrics: (1) exact match (EM), which measures the percentage of correctly predicted answers that exactly match the ground-truth answer; (2) classification F1, which is calculated as the harmonic mean of precision and recall; and (3) accuracy (ACC), which measures the percentage of correctly predicted instances out of all instances. The performance of PT is omitted as it lags far behind other tuning methods in both convergence and performance. The convergence rate of these tuning methods is ranked as: FT > AP ≈ LR > PF.

https://doi.org/10.1038/s42256-023-00626-4

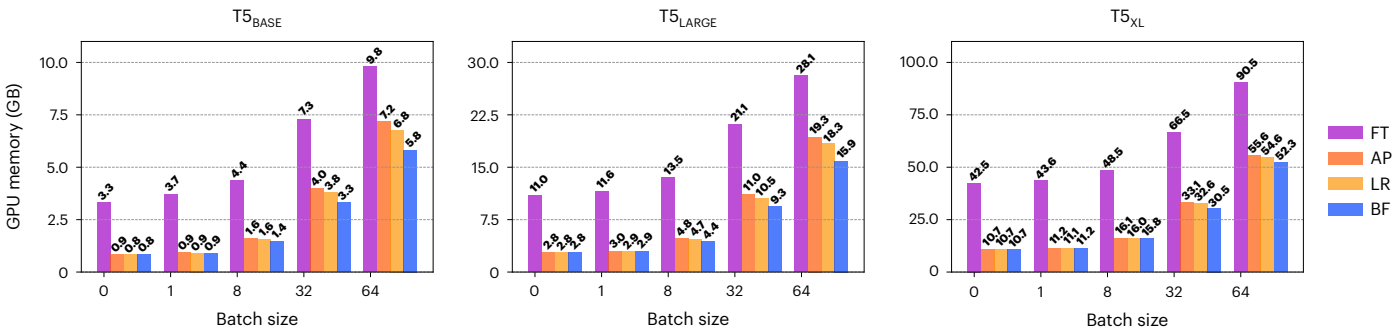


Fig. 2 | GPU memory consumed by each delta-tuning method and fine-tuning. We choose three T5 models with different scales to assess the GPU memory. All evaluations are conducted on NVIDIA A100 GPUs.

6. Transferability. Existing delta-tuning methods could well support knowledge transfer, showing non-trivial transferability among downstream tasks of similar categories. The finding suggests that we could establish a common platform to share and migrate these lightweight delta objects (that is the portion of the fine-tuned parameters).

We discuss the practicality and applications of delta-tuning from various perspectives in Supplementary Section 6, including efficient training and shareable checkpoints, multi-task learning, catastrophic forgetting mitigation and model-as-service. Hopefully, this Analysis will inspire research to advance the efficient adaptation of large language models.

Results

As an effective engine to stimulate large-size PLMs, delta-tuning presents an enormous practical potential for various real-world applications. We carried out systematic experiments to gain a deeper understanding of the attributes of different mainstream delta-tuning methods. Specifically, (1) we first conduct thorough comparisons among four representative delta-tuning methods and fine-tuning, covering the performance, convergence and the efficiency analysis. (2) We explore the combinability of three representative delta-tuning methods by comparing the performance under both the full-data and low-resource settings. We also explore the effects of manual templates and compare the generalization gap of different delta-tuning methods. Furthermore, we investigate (3) the scaling law and (4) the transferability of delta-tuning methods among different downstream tasks. The implementation details and tasks are described in Supplementary Sections 3 and 4.

Performance, convergence and efficiency

Experimental setting. We evaluate vanilla fine-tuning (FT) and four representative delta-tuning methods, including prompt-tuning (PT), prefix-tuning (PF), LoRA (LR) and adapter (AP). We follow the common practice for each delta-tuning implementation, and the training details are provided in Supplementary Section 3.1.

To cover broad and diverse NLP tasks, we select over 100 representative tasks from Huggingface datasets[18]. The selected tasks include text classification (for example, sentiment analysis and natural language inference), question answering (for example, machine reading comprehension and multi-choice question answering), conditional generation (for example, summarization and dialogue) and so on. We list the task details of each category in Supplementary Table 4. To handle different tasks with a single text-to-text PLM, we process the input and output of each task into the same sequence-to-sequence format. $T5_{BASE}$ and $T5_{LARGE}$ are two PLMs with the T5 architecture released by ref. [8]. We choose $T5_{BASE}$ (ref. [8]) as the mainly evaluated PLM backbone for different tuning methods, and we also report the performance of PT with $T5_{LARGE}$ (ref. [8]).

Performance analysis. The overall results are listed in Table 1, from which we observe the following.

1. In general, despite the substantial reduction of tunable parameters, different delta-tuning methods are almost comparable to FT in performance in most cases. This demonstrates the potential of driving large-scale PLMs through parameter-efficient adaptation.
2. Despite having different design elements, PF, LR and AP are comparable to each other in performance. Specifically, each can show dominant performance (even better than FT) over others on certain tasks. According to the average results, the performances of all the methods are ranked as FT > LR > AP > PF > PT. Interestingly, the performance of the delta-tuning methods is not consistent with their number of tunable parameters, that is, at least on small PLMs, more tunable parameters do not necessarily lead to better performance, and the design of the structure for delta-tuning may play a greater role.
3. PT lags far behind other delta-tuning methods in most cases, despite being the easiest method to implement (that is, without modifying the internal structure of the model). Another interesting finding is that, better PT performance is observed when the model size is enlarged to $T5_{LARGE}$, which is aligned with previous findings on the power of scale for prompt-tuning[19]. However, as we show later, other delta-tuning methods also exhibit far better performance when the scale of the backbone PLM grows extremely large. The phenomenon implies that when the model size increases sharply, the design of the structure may become less important for delta-tuning methods.

Convergence analysis. In Fig. 1, Extended Data Fig. 1 and Supplementary Fig. 3, we visualize the performance of different delta-tuning methods (LR, AP and PF) and fine-tuning (FT) at different training steps to compare their convergence rate. We also report the convergence rate with respect to training time in Extended Data Fig. 2. As PT lags far behind other tuning methods in convergence, we do not visualize it in the figures. However, as mentioned in Methods, PT is the easiest method to implement and it is the desirable method to theoretically and empirically study the convergence issue across different sizes of PLMs. Our findings are summarized as follows.

1. The convergence rate of these tuning methods is ranked as: FT > AP ≈ LR > PF. Overall, FT converges the fastest.
2. We also find empirically that, (1) within a reasonably broad range, the performance and convergence of each delta-tuning method are not sensitive to the number of tunable parameters, but more sensitive to the structures of the methods, and (2) with the scale of PLM growing larger, the convergence of delta-tuning is also accelerated (see 'The power of scale for delta-tuning' section).

To summarize, our experiments yield similar conclusions in convergence and overall performance. These conclusions are

https://doi.org/10.1038/s42256-023-00626-4

Table 2 | Results of combining different delta-tuning methods

Prompt	✗	✗	✗	✗	✓	✓	✓	✓
BitFit	✗	✗	✓	✓	✗	✗	✓	✓
Adapter	✗	✓	✗	✓	✗	✓	✗	✓
Tunable parameters	0%	1.75%	0.09%	1.84%	0.003%	1.76%	0.09%	1.85%
RoBERTa$_{\text{LARGE}}$, full data, without manual templates								
CoLA(Matt.)	4.6	**66.6**$_{1.6}$	63.5$_{0.6}$	65.9$_{0.5}$	42.7$_{2.3}$	63.1$_{1.5}$	63.7$_{0.9}$	64.4$_{0.9}$
SST-2(acc)	50.9	**95.8**$_{0.1}$	95.6$_{0.1}$	95.7$_{0.2}$	95.3$_{0.2}$	95.7$_{0.1}$	95.3$_{0.2}$	95.5$_{0.1}$
MRPC(F1)	1.4	92.7$_{0.2}$	91.9$_{0.4}$	**93.0**$_{0.4}$	85.4$_{0.5}$	92.0$_{0.5}$	92.2$_{0.5}$	92.9$_{0.3}$
STS-B(Pear.)	-6.2	**91.4**$_{0.1}$	90.7$_{0.2}$	90.5$_{0.1}$	83.0$_{2.8}$	90.5$_{0.4}$	90.3$_{0.7}$	90.9$_{0.1}$
QQP(F1.)	6.4	83.5$_{0.1}$	83.5$_{0.0}$	84.4$_{0.0}$	77.2$_{0.4}$	84.3$_{0.0}$	83.6$_{0.1}$	**84.4**$_{0.0}$
MNLI(acc)	34.2	88.6$_{0.2}$	88.0$_{0.2}$	**89.0**$_{0.1}$	77.9$_{2.5}$	88.9$_{0.1}$	88.0$_{0.2}$	88.9$_{0.1}$
QNLI(acc)	50.6	93.7$_{0.3}$	93.4$_{0.3}$	94.2$_{0.1}$	86.2$_{0.5}$	94.2$_{0.1}$	93.2$_{0.3}$	**94.4**$_{0.1}$
RTE(acc)	47.7	**86.8**$_{0.5}$	86.2$_{1.0}$	84.5$_{0.5}$	74.4$_{0.5}$	84.1$_{0.8}$	85.7$_{1.5}$	84.7$_{1.1}$
Average	23.7	**87.4**$_{0.4}$	86.6$_{0.4}$	87.1$_{0.2}$	77.7$_{1.2}$	86.6$_{0.4}$	86.5$_{0.6}$	87.0$_{0.3}$
RoBERTa$_{\text{LARGE}}$, full data, with manual templates								
CoLA(Matt.)	2.2	**66.9**$_{1.1}$	64.2$_{0.5}$	65.5$_{1.0}$	37.8$_{20.8}$	64.7$_{1.3}$	64.8$_{0.7}$	64.9$_{1.0}$
SST-2(acc)	83.6	**96.3**$_{0.2}$	96.1$_{0.1}$	96.2$_{0.2}$	95.7$_{0.2}$	95.8$_{0.1}$	95.9$_{0.1}$	95.8$_{0.2}$
MRPC(F1)	61.9	92.2$_{0.4}$	**92.7**$_{0.6}$	92.7$_{0.2}$	84.2$_{0.5}$	91.8$_{0.2}$	92.2$_{0.4}$	92.0$_{0.4}$
STS-B(Pear.)	-3.3	91.3$_{0.5}$	90.9$_{0.1}$	90.7$_{0.2}$	79.6$_{1.3}$	**91.9**$_{0.3}$	90.8$_{0.4}$	90.1$_{0.6}$
QQP(F1)	49.7	83.6$_{0.1}$	83.6$_{0.0}$	**84.6**$_{0.1}$	77.0$_{0.7}$	84.3$_{0.0}$	83.7$_{0.0}$	84.4$_{0.2}$
MNLI(acc)	50.9	88.6$_{0.1}$	87.7$_{0.1}$	88.7$_{0.1}$	80.2$_{0.2}$	88.7$_{0.1}$	88.0$_{0.1}$	**88.9**$_{0.1}$
QNLI(acc)	50.8	93.6$_{0.1}$	93.1$_{0.2}$	**93.8**$_{0.1}$	86.6$_{0.4}$	93.8$_{0.1}$	93.0$_{0.1}$	93.8$_{0.1}$
RTE(acc)	51.3	**86.9**$_{0.2}$	86.2$_{1.0}$	86.0$_{0.7}$	78.3$_{0.3}$	84.6$_{0.5}$	86.4$_{1.5}$	84.7$_{0.9}$
Average	43.4	**87.4**$_{0.3}$	86.8$_{0.3}$	87.3$_{0.3}$	77.4$_{3.0}$	86.9$_{0.3}$	86.9$_{0.4}$	86.8$_{0.4}$
RoBERTa$_{\text{LARGE}}$, 16 shot, without manual templates								
CoLA(Matt.)	4.6	19.6$_{9.6}$	15.1$_{17.0}$	17.7$_{11.4}$	3.5$_{0.6}$	21.4$_{11.5}$	20.8$_{19.6}$	**21.5**$_{13.4}$
SST-2(acc)	50.9	92.7$_{0.4}$	92.7$_{0.6}$	**93.1**$_{0.6}$	74.9$_{0.6}$	91.7$_{0.8}$	92.2$_{0.5}$	91.6$_{0.7}$
MRPC(F1)	1.4	78.2$_{4.4}$	69.8$_{1.6}$	**81.2**$_{0.0}$	6.2$_{4.1}$	74.6$_{7.1}$	69.3$_{6.5}$	77.4$_{5.4}$
STS-B(Pear.)	-6.2	66.5$_{2.5}$	67.5$_{8.0}$	**71.0**$_{2.5}$	10.7$_{3.5}$	63.3$_{1.6}$	64.7$_{5.6}$	69.6$_{8.6}$
QQP(F1)	6.4	55.9$_{5.8}$	55.1$_{6.8}$	54.6$_{4.2}$	52.4$_{1.4}$	58.3$_{7.2}$	55.1$_{4.8}$	**58.5**$_{6.1}$
MNLI(acc)	34.2	58.1$_{4.5}$	**64.6**$_{3.4}$	62.7$_{4.1}$	35.3$_{0.6}$	61.4$_{3.9}$	61.4$_{5.1}$	61.0$_{3.8}$
QNLI(acc)	50.6	60.2$_{3.0}$	**69.7**$_{1.9}$	59.8$_{1.7}$	52.8$_{1.0}$	60.2$_{4.9}$	60.9$_{4.0}$	61.6$_{7.0}$
RTE(acc)	47.7	55.0$_{1.6}$	54.5$_{0.8}$	54.9$_{2.9}$	50.1$_{0.7}$	58.2$_{2.5}$	54.6$_{2.4}$	**58.7**$_{3.4}$
Average	23.7	60.8$_{4.0}$	61.1$_{5.0}$	61.9$_{3.4}$	35.7$_{1.6}$	61.2$_{4.9}$	59.9$_{6.1}$	**62.5**$_{6.0}$
RoBERTa$_{\text{LARGE}}$, 16 shot, with manual templates								
CoLA(Matt.)	2.2	**10.5**$_{15.0}$	4.6$_{5.0}$	9.2$_{10.2}$	1.4$_{1.7}$	10.2$_{4.2}$	5.9$_{2.5}$	5.9$_{5.5}$
SST-2(acc)	83.6	**93.1**$_{0.3}$	92.9$_{0.1}$	92.1$_{0.1}$	90.9$_{0.6}$	91.9$_{0.4}$	92.0$_{0.4}$	92.2$_{0.6}$
MRPC(F1)	61.9	77.2$_{1.4}$	74.5$_{4.9}$	**81.2**$_{0.0}$	72.1$_{4.4}$	76.8$_{1.3}$	76.1$_{2.4}$	**81.2**$_{0.0}$
STS-B(Pear.)	-3.3	65.8$_{4.7}$	69.3$_{6.0}$	71.0$_{4.1}$	12.0$_{8.0}$	61.7$_{5.7}$	**71.3**$_{6.4}$	67.1$_{2.8}$
QQP(F1)	49.7	66.6$_{0.5}$	67.8$_{0.5}$	66.3$_{4.1}$	53.4$_{1.0}$	66.9$_{1.9}$	**68.6**$_{1.2}$	67.1$_{2.9}$
MNLI(acc)	50.9	68.0$_{1.4}$	**69.4**$_{3.3}$	68.9$_{0.4}$	53.2$_{2.5}$	67.1$_{1.8}$	67.1$_{2.0}$	68.1$_{0.3}$
QNLI(acc)	50.8	69.5$_{1.1}$	70.2$_{3.4}$	68.1$_{2.4}$	59.4$_{0.5}$	69.9$_{2.5}$	**72.5**$_{3.9}$	70.4$_{2.3}$
RTE(acc)	51.3	70.6$_{3.6}$	67.3$_{5.1}$	**73.0**$_{2.0}$	56.3$_{4.6}$	70.4$_{2.3}$	69.2$_{3.5}$	72.4$_{2.8}$
Average	43.4	65.2$_{3.5}$	64.5$_{3.5}$	**66.2**$_{2.9}$	49.8$_{2.9}$	64.4$_{2.5}$	65.3$_{2.8}$	65.6$_{2.2}$

Performance of RoBERTa$_{\text{LARGE}}$ on GLUE datasets. We report the average result of multiple random seeds on the validation set. A tick symbol denotes that the component is included in the combination and a cross symbol denotes that it is excluded in the combination. The best performance of each dataset is highlighted in bold.

well supported by the fact that we used the same experimental and implementation set-up, the same model selection strategy and diverse tasks.

Efficiency analysis. Here we study the efficiency of delta-tuning from the perspectives of memory efficiency and computation efficiency. For memory efficiency, to validate the efficiency of graphics processing

https://doi.org/10.1038/s42256-023-00626-4

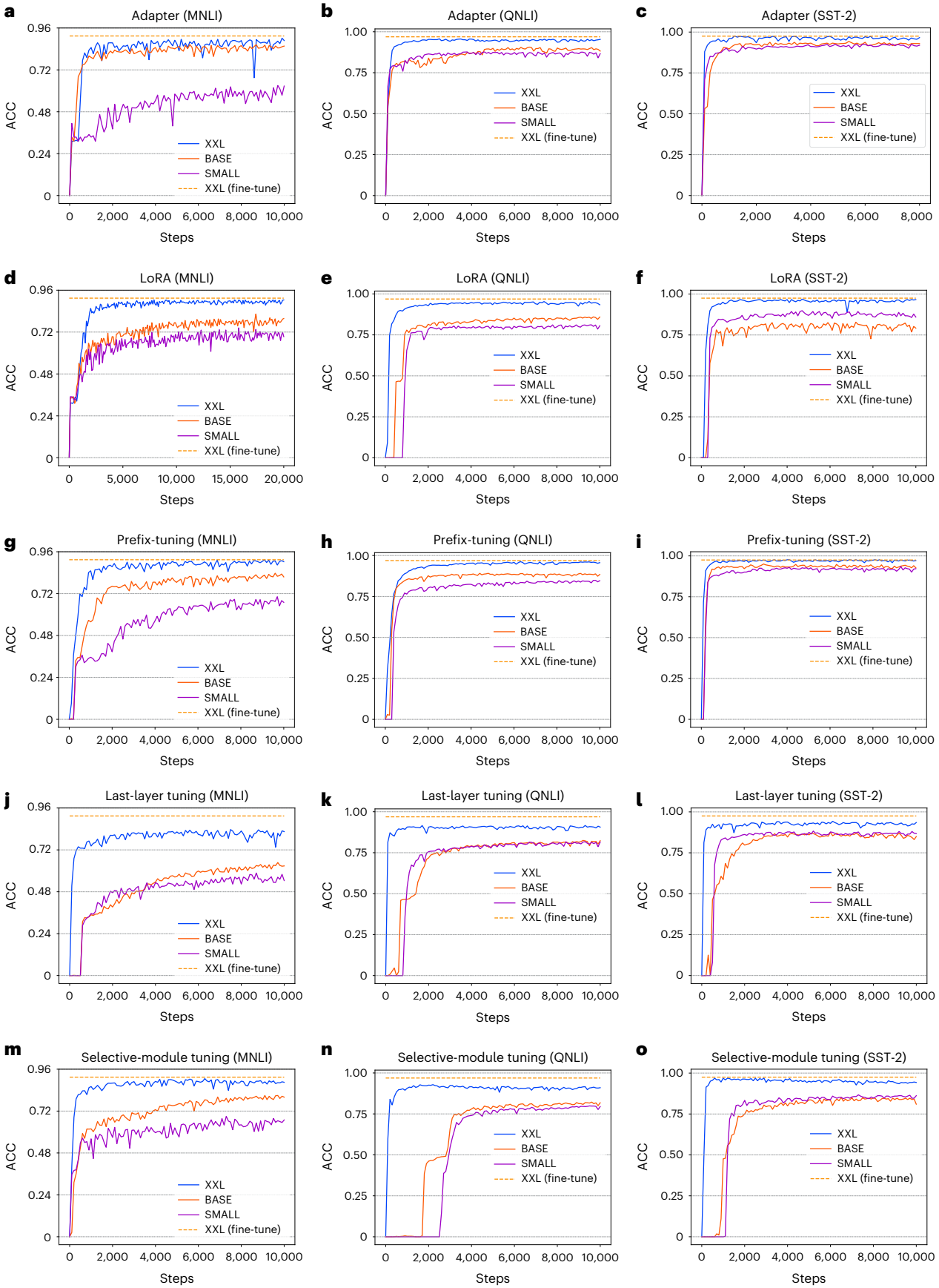


Fig. 3 | The power of scale of delta-tuning methods. a–o, We perform all delta-tuning methods on different scales of T5: $T5_{SMALL}$(), $T5_{BASE}$() and $T5_{XXL}$(). We report the performance of Adapter in (**a–c**), LoRA in (**d–f**), Prefix-tuning in (**g–i**), Last-layer tuning in (**j–l**), and Selective-module tuning in (**m–o**). From this figure, we can observe that with the scale of T5 increasing, all delta-tuning methods could converge faster and achieve better performance on MNLI, QNLI and SST-2.

https://doi.org/10.1038/s42256-023-00626-4

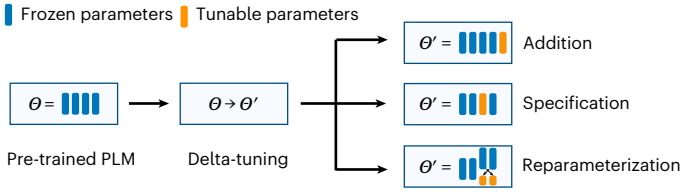


Fig. 4 | The categorization criterion of delta-tuning. Here Θ denotes the pre-trained parameters and Θ' represents the well-tuned parameters.

unit (GPU) memory for delta-tuning, in Fig. 2, we conduct experiments to compare the GPU memory consumed by different delta-tuning methods and fine-tuning across different PLM scales. $T5_{XL}$ is the PLM with the T5 architecture released by ref. [8]. Specifically, we choose three scales of the T5 model, that is, $T5_{BASE}$, $T5_{LARGE}$ and $T5_{XL}$, and test the peak GPU memories under different batch sizes. The static GPU memories, which leave out the intermediate tensors such as hidden states, are drawn on Batchsize=0. We use a NVIDIA A100 GPU (maximum GPU memory 39.58 GB) and library OpenDelta for these experiments. For the cases that consume more GPU memory than a single A100, we parallelize the model across multiple GPUs, which does not introduce additional memory consumption. We observe from the figure that under small batch sizes (for example, 1 and 8), delta-tuning saves up to 3/4 GPU memory; under large batch sizes (for example, 32 and 64), delta-tuning saves about 1/2–1/3 GPU memory. This demonstrates that delta-tuning saves GPU memory by alleviating the need for gradient computations for most of the parameters. Given the fact that small batch sizes are preferred when utilizing big models, delta-tuning has great potential to apply to large-scale PLMs. Furthermore, among the investigated methods, BitFit is the most memory efficient.

In addition, although delta-tuning may converge slower than traditional fine-tuning, the computations of the tunable parameters in the optimizer are greatly reduced, which speeds up training. We compare the forwards time and the backwards time of prompt-tuning, BitFit, adapter tuning and fine-tuning in Extended Data Fig. 3, varying the input length. For a fair comparison, we keep the batch size the same. From the results, we can see that:

1. The structure of the delta-tuning methods could have a considerable impact on the time of a single forwards or backwards process. By greatly reducing the computations of the tunable parameters, the backwards time of delta-tuning methods is shorter than fine-tuning.
2. As the adapter injects additional neural modules to each layer of the transformer model, the path of data flow becomes longer and further leads to inference latency (longer forwards time).

Combinations of delta-tuning methods

Considering that different delta-tuning methods are compatible with each other, which means they could be applied on the same PLM together, we investigate whether such a combination would bring additional benefits. Specifically, we evaluate both simultaneous combination and sequential combination. We choose three representative delta-tuning methods, including prompt-tuning, BitFit and adapter, to explore the effects of their combinations. The training details are described in Supplementary Section 3.2.

Simultaneous combination. We first explore the effects of directly applying all the three delta-tuning methods simultaneously. $RoBERTa_{LARGE}$ is the PLM released by ref. [20] and GLUE[21] is the official benchmark for language understanding ability evaluation. The experiments are conducted using $RoBERTa_{LARGE}$ on eight tasks of GLUE (full-data setting), and we report the performance on the official development sets. We also test the performance of $RoBERTa_{LARGE}$ under the few-shot setting, where we randomly sample 16 training examples per label to construct the new training set and development set, respectively. Similar to prompt-based fine-tuning[22], we insert a natural language prompt template into the input text for each task, and the detailed implementations are described in Supplementary Section 3.2.

We list the results of simultaneous combination for $RoBERTa_{LARGE}$ in Table 2 (the results of $T5_{BASE}$ are listed in Extended Data Table 2, with discussions in Supplementary Section 3.2), from which we conclude that:

1. Under both the full-data setting and few-shot setting, introducing adapter into the combination almost always conduces to the average performance across GLUE tasks no matter whether there exist manual templates.
2. Introducing prompt-tuning into the combination generally harms the average performance, showing that prompt-tuning may not be compatible with the other two delta-tuning methods.
3. Introducing BitFit into the combination generally improves the average performance.
4. Manual templates could substantially improve the zero-shot performance (from 23.7 to 43.4) by narrowing the gap between downstream tuning and pre-training. Under the few-shot setting, manual templates could also help boost the average performance evidently. However, when the training supervision is abundant (full-data setting), manual templates only show marginal improvements.

Sequential combination. In addition to the simultaneous combination, we further investigate the compatibility when the above three delta-tuning methods (prompt-tuning, BitFit and adapter) are sequentially introduced. Specifically, we split the whole tuning process into three stages. During each stage, we train an individual delta-tuning method for 6,000 steps; in the following stages, we freeze the tuned parameters in the previous stages and optimize only the newly introduced delta parameters. SST-2 (ref. [23]) is the dataset that evaluates the sentiment analysis ability. We experiment with $RoBERTa_{LARGE}$ on SST-2 with and without manual templates. The results are visualized in Extended Data Fig. 4, from which it is derived that:

1. Under certain cases, the performance can be improved with the involvement of subsequent delta-tuning methods.
2. However, there does not exist an optimal sequential combination strategy that could dominate other combination strategies under different settings.

Generalization gap. In addition, we report the generalization gap (train performance − dev performance) for $RoBERTa_{LARGE}$ under the full-data setting, with the results shown in Extended Data Table 3. It is derived that:

1. The gap of a single delta-tuning method is always smaller than fine-tuning, which means over-parameterization may help better memorize (overfit) training samples. Among all the delta-tuning methods, prompt-tuning tends to have the smallest generalization gap. Considering that each delta-tuning method could already generalize well and achieve non-trivial performance on the development set, overfitting the training set may not be the prerequisite for good generalization.
2. In general, combining delta-tuning methods would enlarge the generalization gap, even to the extent that is comparable to fine-tuning, despite tuning far fewer parameters. This suggests that, for the investigated tasks, memorizing the training set may not require employing all of the parameters; in other words, a small model capacity during downstream adaptation may be enough for good memorization.
3. Utilizing manual templates generally would not influence the generalization gap.

Conclusion. The above experiments indicate that different delta-tuning methods have distinct functionalities for the optimization

of PLMs; thus, combining them is generally conducive to the downstream performance. However, as shown in the above results, the optimal combination of delta-tuning methods may vary considerably under different settings. That being said, it would be interesting to explore the mechanisms behind the inductive biases brought by different delta-tuning methods under different cases in the future. Besides, we also encourage future research explorations to systematically report the performance of their proposed delta-tuning methods on various PLM backbones under different settings thoroughly.

The power of scale for delta-tuning

With the scale of the backbone PLM growing, prompt-tuning becomes more and more competitive in performance, and would even achieve comparable performance to fine-tuning for a PLM with over 10 billion parameters[19], and the convergence speed of prompt-tuning benefits from the scaling law. In this section, we explore whether other delta-tuning methods also exhibit the power of scale. MNLI and QNLI are two natural language inference dataset, and $T5_{SMALL}$ and $T5_{XXL}$ are two PLMs with the T5 architecture released by ref. [8]. Specifically, we experiment on the task of MNLI, QNLI and SST-2, and choose three PLMs ($T5_{SMALL}$, $T5_{BASE}$ and $T5_{XXL}$) of increasing sizes, and evaluate the performance of five representative delta-tuning methods (adapter, LoRA, prefix-tuning, last-layer tuning and selective-module tuning). We describe the percentages of the tuned parameters for each method in all scales of the PLM in Supplementary Table 3. The training details are provided in Supplementary Section 3.3. The results are visualized in Fig. 3. From Fig. 3a–i, we observe that with the scale of the PLM growing, both the performance and the convergence of all delta-tuning methods are greatly improved. All delta-tuning methods tend to show comparable performance to fine-tuning, even for a small-scale PLM ($T5_{BASE}$).

On the basis of the existing results, we further design two delta-tuning methods: last-layer tuning and selective-module tuning. For last-layer tuning, we optimize the last layer in the T5 encoder; for selective-module tuning, we randomly choose some modules (for example, the feed-forward layer, query/key/value matrix in the attention layer, or a layer norm) in the T5 model to be tunable. The results are visualized in Fig. 3j–l,m–o, from which we could conclude that:

1. Both methods show promising results, especially when the scale of the PLM is extremely large, with selective-module tuning slightly better than last-layer tuning. These results suggest that confining the optimization within a specific layer may not be a good strategy (for example, the case of prompt-tuning and last-layer tuning).
2. Furthermore, randomly choosing modules across different layers could achieve excellent performance when the scale of PLMs grows extremely large.

In general, the above results imply that the power of scale may be a common phenomenon for delta-tuning. We hypothesize the existence of such a phenomenon is because larger PLMs generally have smaller intrinsic dimensionalities[16]; therefore, merely tuning minimal parameters could obtain a strong enough representation ability to achieve non-trivial performance in downstream tasks; furthermore, the over-parameterization and large-scale pre-training may make PLMs more unlikely to get stuck in a local optimum during downstream optimization, and thus the convergence is accelerated.

Task-level transferability evaluation

Recent studies[24–26] have demonstrated that prompt-tuning has excellent cross-task transferability. In this subsection, we explore the cross-task transferability of four delta-tuning methods (prompt-tuning, prefix-tuning, adapter and LoRA) with 12 tasks of 5 different types (sentiment analysis, natural language inference, paraphrase identification, question answering and summarization). We transfer the trained delta parameters to the unseen target tasks. More training and dataset details are provided in Supplementary Section 3.4.

In experiments, we report their relative performance (zero-shot transferring performance and original performance). The results are shown in Extended Data Fig. 5, from which we can observe that:

1. For the tasks belonging to the same category, transferring tuned parameters among them generally performs well; for the tasks of different types, transferring delta parameters among them generally achieves poor performance.
2. We also find that transferring tuned parameters from the text generation tasks such as question answering and summarization can achieve non-trivial performance on sentiment analysis, indicating that text generation might be a complex task that includes the knowledge required to solve the sentiment analysis tasks. In general, the above results demonstrate that it is promising to utilize trained delta parameters for similar tasks through knowledge transfer.

Conclusion

This Analysis focuses on parameter-efficient methods, that is, delta-tuning, for PLMs. We first describe the problem and provide a categorization to survey the development of delta-tuning systematically. Captivated by the empirical evidence, we propose two frameworks to theoretically discuss delta-tuning from the optimization and optimal control perspectives. Our discussion sheds light on the theoretical references of a novel design for delta-tuning methods and hopefully could inspire a deeper understanding of model adaptation for PLMs. Empirically, we conduct extensive experiments across 100+ NLP tasks to fairly evaluate and explore the combinatorial property, influence of scale and transferability for delta-tuning. In terms of performance, delta-tuning can be slightly behind or comparable to fine-tuning on a wide range of tasks, and the gap shrinks as the model scales; in terms of efficiency, delta-tuning could considerably reduce storage space and memory usage, as well as accelerate backpropagation. In summary, delta-tuning shows considerable potential to stimulate large PLMs, and we hope that the paradigm can be further theoretically studied and empirically practiced.

Methods

Delta-tuning is developed on the success of PLMs, which use deep transformers as the base structure and adopts pre-training objectives on large-scale unlabelled corpora. For more information about PLMs and transformers, see Supplementary Section 1 or related surveys[27] and original papers[4,5,8,9].

Given a pre-trained model $\Theta = \{w_1, w_2, ..., w_N\}$ and training data $\mathcal{D}$, the objective of PLM adaptation is to produce the adapted model $\Theta' = \{w'_1, w'_2, ..., w'_M\}$, where w_i is the model parameter. Define $\Delta\Theta$ as the change in the adapted model Θ' compared with Θ, including the change in values and the number of elements. In vanilla fine-tuning, $N = M$ and $\Delta\Theta = \nabla f_\Theta(\mathcal{D})$ is the update value of all parameters in Θ with respect to training data, where f_Θ represents the resulting loss of applying model Θ to training data D. Note that in this case, we omit the small set of parameters brought by extra classification heads for downstream tasks. While in delta-tuning, $\Delta\Theta$ refers to the modification of a small number of parameters. Empirically, $|\Delta\Theta| = |\Theta|$ in vanilla fine-tuning, while for delta-tuning, $|\Delta\Theta| \ll |\Theta|$, where $|\cdot|$ indicates the number of parameters involved.

To organize them under a unified framework, we categorize the delta-tuning methods into three groups according to the operations on the delta parameters (as illustrated in Fig. 4): addition-based, specification-based and reparameterization-based approaches.

- Addition-based methods introduce extra trainable neural modules or parameters that do not exist in the original model or process. In addition-based methods, $M \geq N$ and $\Delta\Theta = \{w_{N+1}, w_{N+2}, ..., w_M\}$.

- Specification-based methods specify certain parameters in the original model or process become trainable, whereas others are frozen. Denote the set of trainable parameters as $\mathcal{W}$, then $\Delta\Theta = \{\Delta w_1, \Delta w_2, ..., \Delta w_N\}$. When $w_i \in \mathcal{W}$, Δw_i is the incremental value from w_i to w_i', else, $\Delta w_i = 0$.
- Reparameterization-based methods reparameterize existing parameters to a parameter-efficient form by transformation. Denote the set of parameters to be reparameterized as $\mathcal{W}$, and suppose that each $w_i \in \mathcal{W}$ is reparameterized with new parameters $R(w_i) = \{u_1, u_2, ..., u_{N_i}\}$, then $\Delta\Theta = (\Theta \setminus \mathcal{W}) \cup \mathcal{U}$, where $\mathcal{U} = \{u_j | \exists w_i \in \mathcal{W}, u_j \in R(w_i)\}$.

Addition-based methods

With the above definition in mind, addition-based methods introduce additional parameters to the neural network. In this section, we introduce two branches of representative addition-based methods, adapter-based tuning and prompt-based tuning.

Adapter-based tuning. As a seminal work in delta-tuning, adapter-based methods inject small-scale neural modules (adapters) to the transformer layers and only tune these adapters for model adaptation. Although such a strategy leaves an open choice of adapter structures, a simple instantiation[13] achieves impressive performance and has become the most widely used baseline in recent research. Specifically, one adapter module contains a down-projection and an up-projection. For an input feature $\mathbf{h} \in \mathbb{R}^d$, a down-projection projects the input to a r-dimensional space with a parameter matrix $W_d \in \mathbb{R}^{d \times r}$, after which a nonlinear function $f(\cdot)$ is applied. Then the up-projection W_u maps the r-dimensional representation back to d-dimensional space. Added with a residual connection, the complete computation could be written as $\mathbf{h} \leftarrow f(\mathbf{h}W_d)W_u + \mathbf{h}$.

In each block, the adapter modules are separately inserted after the multi-head self-attention and the feed-forward network sublayers, which reduces the tunable parameters per layer to $2 \times (2dr$ (projection-matrices) $+ d$ (residualconnection) $+ r$ (biasterm)). Practically, about 0.5–8% of parameters of the whole model[13] could be involved in the tuning process under such a strategy.

Although an adapter works with much fewer tunable parameters than vanilla fine-tuning, some work attempts a more rigorous saving strategy by introducing inductive biases into the structure of the adapter layer. For example, Compacter[28] proposes to use a combination of hypercomplex multiplication and parameter sharing. The hypercomplex multiplication parameterizes the original linear layer as the sum of the Kronecker products of two small matrices. Taking the down-projection as an example, $W_d = \sum_{i=1}^{n} A_i \otimes B_i$, where $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{\frac{d}{n} \times \frac{r}{n}}$.

Their method reduces the parameter complexity of the normal adapter layer from $\mathcal{O}(dr)$ to $\mathcal{O}(d + r)$ without harming the performance. It also shows that a simple low-rank decomposition of the linear layer leads to comparable performance with the adapter layer, that is, $W_d = AB^T$, where $A \in \mathbb{R}^{d \times n}$, $B \in \mathbb{R}^{r \times n}$ and $n \ll \min(d, r)$, where the superscript T means matrix transposition.

As an addition-based approach, adapter-based tuning has the advantage of placing multiple adapter instances on a pre-trained model simultaneously, which can benefit many application scenarios. For example, multi-task learning[29,30] is an advantageous setting for adapter-based methods, inserted with adapter modules in parallel with the self-attention module, PLMs could demonstrate impressive representational capacity in the multi-task setting. In contrast to directly conducting multi-task learning on adapters, adapterFusion[31] first pre-trains task-specific adapters and then combines the representations of the pre-trained adapters to leverage the cross-task knowledge and enhance the performance of transfer learning.

In terms of computational efficiency, the training of adapters could be 60% faster than vanilla fine-tuning while the inference is only 4–6% slower. In addition, the computational cost could be further reduced dynamically by removing adapters from lower transformer layers[32]. Research also shows that adapter-based fine-tuning demonstrates better robustness than fine-tuning. Specifically, adapter-based fine-tuning could perform better than vanilla fine-tuning on few-shot and cross-lingual scenarios[33] and is more robust under adversarial attacking[34]. We provide a comparison of different adapters, as well as other delta-tuning methods in Extended Data Table 4.

To sum up, adapters are lightweight additional neural modules that could be trained in a task-specific style, which could be regarded as 'encapsulation' of task information (in fact, this perspective can be applied to all the 'deltas'). Although in an ideal world, adapters could be freely shared and reused by researchers, in practice, sharing and reusing such modules face substantial obstacles. Taking the first step, AdapterHub[35] provides a feasible platform and toolkit to deploy adapters inside the transformer-based models.

Prompt-based tuning. Instead of injecting neural modules to the transformer model, prompt-based methods wrap the original input with additional context. As a strategy to stimulate PLMs by mimicking pre-trained objectives in the downstream tasks, prompt-based learning has achieved promising performance in various NLP tasks[36,37], especially in low-data settings. The introduction of the technique and implementations of prompt-based learning have already been comprehensively presented in other literature[38,39]. In this paper, we primarily focus on the parameter-efficient attribute of prompt-based learning (only prefixes or prompts are optimized) and pay less attention to the settings where the models and prompts are simultaneously optimized.

An important seminal work of this branch of research is prefix-tuning[40], which prepends trainable continuous tokens (prefixes) to the input and hidden states of each transformer layer. Each prefix is drawn from a newly initialized trainable parameter matrix $\mathbf{P}$, whereas other parameters of the pre-trained model remain unchanged during training. During generation, if an activation h_i is in a prefix position, it is the direct copy of the corresponding trainable parameter; otherwise, the activation is computed by the model as $h_i = \mathrm{LM}(z_i, h_{<i})$, where i is the position index, z is the input and LM stands for the language model. It is worth noting that the paradigm could be applied to both autoregressive and encoder–decoder models. Such a strategy could be effectively applied to natural language understanding with different scales of models[41].

Compared with prefix-tuning, which adds tunable prefixes to every intermediate transformer layer, prompt-tuning[19] proposes a more simplified strategy that only adds soft prompts to the input layer. Similar to prefix-tuning, the newly introduced prompts are not parameterized by the pre-trained model but an additional parameter matrix. And during training, the parameters of soft prompts are updated by gradient descent while the model parameters keep frozen. As the model size increases, the performance gap between prompt-tuning and full parameter fine-tuning is narrowed. In particular, when the model scales to $\mathrm{T5}_{XXL}$ with 11 billion parameters, prompt-tuning yields comparable performance on SuperGlue with fine-tuning. This strategy also exhibits sensitivity to the length and initialization of the soft prompts. Prompts could also be injected in the pre-training stage to seek a satisfying initialization point[42]. Moreover, similar to other methods, prompt-tuning also demonstrates transferability across tasks[24,26], which suggests that appropriate initialization could be substantially beneficial for downstream tasks.

The training curse of prompt-based methods. Although prompt-based methods exhibit a promising future for the adaptation of large pre-trained models, especially as prompt-tuning does not need to modify anything inside the neural network, there still exist unsolved challenges. In practice, prompt-tuning is difficult to optimize, and generally, this phenomenon becomes more apparent as the volume

of data and the size of the model decreases. Even though soft prompts can be trained successfully, they converge slower than full parameter fine-tuning and other delta-tuning methods during training. In our experiments, we validate the phenomenon across different datasets ('Performance, convergence and efficiency' section), indicating that it is an interesting topic to train soft prompts to converge stably in various situations.

Specification-based methods

Specification-based methods fine-tune a few inherent parameters while leaving the majority of parameters unchanged in model adaptation. This approach does not seek to change the internal structure of a model but to optimize a small number of internal parameters to solve particular tasks. In general, such specifications could be implemented based on heuristics or training supervision.

Heuristic specification. Specification-based methods do not introduce any new parameters to the model, but directly specify part of the parameters to be optimized. The idea is simple but surprisingly effective; an early study[43] only fine-tunes one-fourth of the final layers of BERT and RoBERTa and could produce 90% of the performance of full parameter fine-tuning. BitFit[14] empirically proves that by only optimizing the bias terms inside the model and freezing other parameters, the model could still reproduce over 95% performance on several benchmarks. Empirical results in BitFit also show that even if we use a small random set of parameters for delta-tuning (which obviously will degrade the performance), the model could still yield passable results on the GLUE benchmark. Unfortunately, the work only applies this trick to small-scale models, and there is no guarantee that randomly choosing some parameters to be tuned would remain competitive for larger models. Another valuable observation is that different bias terms may have different functionalities during model adaptation.

Learn the specification. Rather than manually or heuristically specify which parameters to update, one alternative is to 'learn' such specifications. Following the definition in this section, diff pruning[44] reparameterizes the fine-tuned model parameters Θ' as the summation of the pre-trained parameters Θ and the difference vector $\Delta\Theta$, that is, $\Theta' = \Theta + \Delta\Theta$, where $|\Theta| = |\Theta'|$. Hence, the key issue is to encourage the difference vector to be as sparse as possible; this work regularizes the vector by a differentiable approximation to the L_0-norm penalty to achieve the goal of sparsity. Practically, because new parameters to be optimized are introduced in the learning phase, diff pruning takes up more GPU memory than full parameter fine-tuning, which may establish barriers in the application on large PLMs. The masking method[45] learns selective masks for PLMs to only update the critical weights for particular tasks. To learn such a set of masks, a binary matrix associated with the model weights is introduced, where each value is generated by a thresholding function. During backpropagation, the matrix is updated by a noisy estimator.

Reparameterization-based methods

Reparameterization-based methods transform the adaptive parameters during optimization into parameter-efficient forms. This branch of delta-tuning is typically motivated by the hypothesis that PLM adaptations towards most downstream tasks are inherently low rank, and could thus be equivalently completed in a parameter-efficient way.

Intrinsic dimensions of PLM adaptation. Previous work[16] has empirically shown that the full parameter fine-tuning process of pre-trained models can be reparameterized into optimization within a low-dimensional subspace, that is, fine-tuning has a low intrinsic dimension[46], which measures the minimum number of parameters needed to reach satisfactory performance. In experiments, they find that a relatively low-dimensional (for example, thousands) reparameterization could achieve over 85% fine-tuning performance. In this sense, PLMs may serve as general compression frameworks, which compress the optimization complexity from high dimensions to low dimensions. They also demonstrate that larger PLMs generally have smaller intrinsic dimensions, and the process of pre-training implicitly reduces the PLM's intrinsic dimension. Taking inspiration from these observations, reparameterization-based delta-tuning methods are proposed, which reparameterize (a part of) original model parameters with low-dimensional proxy parameters and only optimize the proxy parameters and thus reduce the computation and memory cost.

Intrinsic rank of weight differences. LoRA[15] hypothesizes that the change of weights during model tuning has a low intrinsic rank. On the basis of this hypothesis, it is proposed to optimize the low-rank decomposition for the change of original weight matrices in the self-attention modules. In deployment, the optimized low-rank decomposition matrices are multiplied to obtain the delta of self-attention weight matrices. In this way, LoRA could match the fine-tuning performance on the GLUE benchmark. They demonstrate the effectiveness of their methods on PLMs of various scales and architectures.

Intrinsic space of multiple adaptations. Furthermore, intrinsic prompt-tuning[17] makes a stronger hypothesis that the adaptations to multiple tasks could be reparameterized into optimizations within the same low-dimensional intrinsic subspace. Instead of resorting to a random subspace[16], they try to find a common subspace shared by various NLP tasks, which is implemented through decomposing the trained soft prompts of multiple NLP tasks into the same low-dimensional nonlinear subspace, and then learn to adapt the PLM to unseen tasks or data by only tuning parameters in the subspace. Experiments show that in a 250-dimensional subspace found with 100 random tasks, by only tuning 250 free parameters, 97% and 83% of the full prompt-tuning performance can be recovered for 100 seen tasks (using different training data) and 20 unseen tasks, respectively. This provides strong evidence for their universal reparameterization hypothesis and may inspire future work. Moreover, this work also shows that the low-dimensional reparameterization can substantially improve the stability of prompt-tuning. Their method could also be leveraged as a tool for analysing the similarity and differences between various NLP tasks.

Theoretical perspectives of delta-tuning

Are these methods essentially doing the same thing? We are interested in the theoretical principles behind delta-tuning. A PLM can usually be effectively adapted to various downstream tasks with a smaller cost compared with pre-training, which leads to theoretical issues that are worth exploring in depth. We adopt two frameworks to introduce theoretical insights into delta-tuning from the perspectives of optimization and optimal control.

Optimization perspective. As training neural networks is an optimization process, the mechanism of delta-tuning can be analysed from the perspective of optimization. In general, it is challenging and time-consuming to solve large-scale and high-dimensional optimization problems. However, in the fine-tuning of a large PLM, empirical study[16] reveals that there exists a low intrinsic dimension; thus, some customized optimization schemes can benefit from this property and be quite efficient in practice. One promising scheme is the subspace optimization[47] that seeks an acceptable solution in a low-dimensional subspace. It manipulates a small number of variables and is more economical than the optimization in the whole space. In fact, delta-tuning can be viewed as a subspace-optimization method.

There are two approaches to applying subspace optimization and thus the delta-tuning can roughly fall into two categories. One is tuning model parameters in the solution subspace. It exploits a low-dimensional manifold that can approximately represent the

whole model parameters, and the optimization trajectory follows this manifold. Some delta-tuning methods can be categorized into this approach, for example, LoRA[15], BitFit[14] and diff pruning[44]. The other approach seeks a surrogate of the original objective function in a small functional subspace and uses the minimizer of the surrogate function as the approximate final solution. It can provide some explanations of the rationales of some popular delta-tuning methods such as prompt-tuning[19] and prefix-tuning[40]. A complete discussion can be found in Supplementary Section 2.1.

Optimal control perspective. We draw inspiration from optimal control theories to better understand the functionality of delta-tuning. In addition to their parameter efficiency, the essence of delta-tuning lies in regularizing the layer-wise hidden-state transformation process along forwards propagation. The forward propagation of hidden states h between layer j and $j+1$ in the PLM, with the guidance of the delta parameters $\delta^{(j)}$ at the jth layer, can be written as $\mathcal{G}_\theta^{(j)}(h^{(j)}, \delta^{(j)})$. With the parameters θ in the PLM fixed, the transformation function $\mathcal{G}_\theta^{(j)}$ defines the altered forwards propagation at the jth layer with the learnable $\delta^{(j)}$. The detailed formulations and instantiations of $\mathcal{G}_\theta^{(j)}$ for different delta-tuning methods, including Prefix-tuning, Adapter, LoRA and BitFit, are listed in Supplementary Section 2.2. In this way, the tuned delta parameters are interpreted as the optimal controllers that steer the PLMs to work in different realistic settings.

The optimal control perspective instructs the novel design of delta-tuning. For example, robust prefix-tuning[48] tunes additional layer-wise prefix parameters during inference. The layer-wise propagation of hidden states is thus guided towards correct outputs. Another work[49] leveraged inference-time bias-term tuning to mitigate bias and toxicity in natural language generation. The number of bias terms to be tuned is determined by the extent of modification of the hidden-state transformation in an adaptive manner. Finally, by applying the theories of controller design[50,51], we expect more delta-tuning methods proposed with theoretical guarantees and better exploitation of the power of PLMs[52].

Data availability

Datasets used in this study are freely available at https://github.com/INK-USC/CrossFit and https://huggingface.co/datasets/glue.

Code availability

The source code of this study is publicly available on GitHub at https://github.com/thunlp/OpenDelta. It is also available at https://zenodo.org/record/7340282.

References

1. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015).
2. Hochreiter, S. & Schmidhuber, J. ürgen. *Long short-term memory. Neural Comput.* **9**, 1735–1780 (1997).
3. Bengio, Y., Ducharme, R. & Vincent, P. A neural probabilistic language model. In *Advances in Neural Information Processing Systems*. 13 (2000).
4. Vaswani, A. et al. Attention is all you need. In *Advances in Neural Information Processing Systems*. 30 (2017).
5. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proc. the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. **1**, 4171–4186 (2019).
6. Radford, A., Narasimhan, K., Salimans, T. & Sutskever, I. Improving language understanding by generative pre-training. *OpenAI Blog*. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf (2018).
7. Radford, A. et al. Language models are unsupervised multitask learners. *OpenAI Blog*. https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf (2019).
8. Raffel, C. et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.* **21**, 5485–5551 (2020).
9. Brown, T. et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*. **33**, 1877–1901 (2020).
10. Rae, J. W. et al. Scaling language models: methods, analysis & insights from training Gopher. Preprint at *arXiv* https://arxiv.org/abs/2112.11446 (2021).
11. Smith, S. et al. Using deepspeed and megatron to train Megatron-Turing NLG 530b, a large-scale generative language model. Preprint at *arXiv* https://arxiv.org/abs/2201.11990 (2022).
12. Chowdhery, A. et al. PaLM: scaling language modeling with pathways. Preprint at *arXiv* https://arxiv.org/abs/2204.02311 (2022).
13. Houlsby, N. et al. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*. (eds Chaudhuri, K. & Salakhutdinov, R.) 2790–2799 (2019).
14. Zaken, E. B., Ravfogel, S. & Goldberg, Y. Bitfit: simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proc. the 60th Annual Meeting of the Association for Computational Linguistics*. **2**, 1–9 (2022).
15. Hu, E. J. et al. LoRA: low-rank adaptation of large language models. In *International Conference on Learning Representations* (2022).
16. AAghajanyan, A., Gupt, S. & Zettlemoyer, L. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proc. the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*. **1**, 7319–7328 (2021).
17. Qin, Y. et al. Exploring low-dimensional intrinsic task subspace via prompt tuning. Preprint at *arXiv* https://arxiv.org/abs/2110.07867 (2021).
18. Lhoest, Q. et al. Datasets: a community library for natural language processing. In *Proc. the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. 175–184 (2021).
19. Lester, B., Al-Rfou, R. & Constant, N. The power of scale for parameter-efficient prompt tuning. In *Proc. the 2021 Conference on Empirical Methods in Natural Language Processing*. *3045–3059* (2021).
20. Liu, Y. et al. Roberta: a robustly optimized BERT pretraining approach. Preprint at *arXiv* https://arxiv.org/abs/1907.11692 (2019).
21. Wang, A. et al. GLUE: a multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations* (2019).
22. Schick, T. & Schütze, H. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proc. the 16th Conference of the European Chapter of the Association for Computational Linguistics*. 255–269 (2021).
23. Socher, R. et al. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proc. the 2013 Conference on Empirical Methods in Natural Language Processing*. 1631–1642 (2013).
24. Su, Y. et al. On transferability of prompt tuning for natural language understanding. In *Proc. the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 3949–3969 (2022).
25. Williams, A., Nangia, N. & Bowman, S. A broad-coverage challenge corpus for sentence understanding through inference. In *Proc. the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 1, 1112–1122 (2018).

26. Vu, T., Lester, B., Constant, N., Al-Rfou, R. & Cer, D. Spot: better frozen model adaptation through soft prompt transfer. In *Proc. the 60th Annual Meeting of the Association for Computational Linguistics*. 1, 5039–5059 (2022).
27. Han, X. et al. Pre-trained models: Past, present and future. *AI Open* **2**, 225-250. https://www.sciencedirect.com/science/article/pii/S2666651021000231 (2021).
28. Mahabadi, R. K., Henderson, J. & Ruder, S. Compacter: efficient low-rank hypercomplex adapter layers. *In Advances in Neural Information Processing Systems*. **34**, 1022–1035 (2021).
29. Stickland, A. C. & Murray, I. BERT and pals: projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning*. 5986–5995 (2019).
30. Mahabadi, R. K., Ruder, S., Dehghani, M. & Henderson, J. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Proc. the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*. 1, 565–576 (2021).
31. Pfeiffer, J., Kamath, A., Rücklé, A., Cho, K. & Gurevych, I. AdapterFusion: non-destructive task composition for transfer learning. In *Proc. the 16th Conference of the European Chapter of the Association for Computational Linguistics*. 487–503 (2021).
32. Rücklé, A. et al. AdapterDrop: in the efficiency of adapters in transformers. In *Proc. the 2021 Conference on Empirical Methods in Natural Language Processing*. 7930–7946 (2021).
33. He, R. et al. On the effectiveness of adapter-based tuning for pretrained language model adaptation. In *Proc. the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*. 1, 2208–2222 (2021).
34. Han, W., Pang, B. & Wu, Y. N. Robust transfer learning with pretrained language models through adapters. In *Proc. the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*. **2**, 854–861 (2021).
35. Pfeiffer, J. et al. AdapterHub: a framework for adapting transformers. In *Proc. the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. 46–54 (2020).
36. Gao, T., Fisch, A. & Chen, D. Making pre-trained language models better few-shot learners. In *Proc. the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*. **1**, 3816–3830 (2021).
37. Hu, S. et al. Knowledgeable prompt-tuning: incorporating knowledge into prompt verbalizer for text classification. In *Proc. the 60th Annual Meeting of the Association for Computational Linguistics*. **1**, 2225–2240 (2021).
38. Liu, P. et al. Pre-train, prompt, and predict: a systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.* **55**, 1–35 (2023).
39. Ding, N. et al. Openprompt: an open-source framework for prompt-learning. In *Proc. the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. 105–113 (2022).
40. Li, X. L. & Liang, P. Prefix-tuning: optimizing continuous prompts for generation. In *Proc. the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*. **1**, 4582–4597 (2021).
41. Liu, X. et al. P-tuning: prompt tuning can be comparable to fine-tuning universally across scales and tasks. In *Proc. the 60th Annual Meeting of the Association for Computational Linguistics*. **2**, 61–68 (2022).
42. Gu, Y., Han, X., Liu, S. & Huang, M. Ppt: pre-trained prompt tuning for few-shot learning. *In Proc. the 60th Annual Meeting of the Association for Computational Linguistics*. **1**, 8410–8423 (2022).
43. Lee, J., Tang, R. & Lin, J. What would elsa do? Freezing layers during transformer fine-tuning. Preprint at *arXiv* https://arxiv.org/abs/1911.03090 (2019).
44. Guo, D., Rush, A. & Kim, Y. Parameter-efficient transfer learning with diff pruning. In *Proc. the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*. **1**, 4884–4896 (2021).
45. Zhao, M., Lin, T., Mi, F., Jaggi, M. & Schütze, H. Masking as an efficient alternative to finetuning for pretrained language models. In *Proc. the 2020 Conference on Empirical Methods in Natural Language Processing*. 2226–2241 (2020).
46. Li, C., Farkhoor, H., Liu, R. & Yosinski, J. Measuring the intrinsic dimension of objective landscapes. In *International Conference on Learning Representations* (2018).
47. Liu, X., Wen, Z. & Yuan, Y.-X. Subspace methods for nonlinear optimization. *CSIAM Trans. Appl. Math.* **2**, 585–651 (2021).
48. Yang, Z. & Liu, Y. On robust prefix-tuning for text classification. In *International Conference on Learning Representations* (2022).
49. Yang, Z., Yi, X., Li, P., Liu, Y. & Xie, X. Unified detoxifying and debiasing in language generation via inference-time adaptive optimization. Preprint at *arXiv* https://arxiv.org/abs/2210.04492 (2022).
50. Boyd, S. P. & Barratt, C. H. *Linear Controller Design: Limits of Performance* Vol. 7 (Citeseer, 1991).
51. Ang, K. H., Chong, G. & Li, Y. PID control system analysis, design, and technology. *IEEE Trans. Control Syst. Technol.* **13**, 559–576 (2005).
52. He, J., Zhou, C., Ma, X., Berg-Kirkpatrick, T. & Neubig, G. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations* (2022).

Acknowledgements

This work is supported by the National Key Research and Development Program of China (No. 2020AAA0106500), National Natural Science Foundation of China (No. 62276154 and No. 62011540405), Beijing Academy of Artificial Intelligence (BAAI) and the Institute for Guo Qiang at Tsinghua University. We thank J. He, P. Liu, T. Sun, C., L. Wang, C. Fang, X. Han and R. Shao for their suggestions and help with the paper.

Author contributions

N.D., Y.Q. and Z.L. initiated and organized the research. N.D. drafted the abstract, the main text and Section Methods. S.H., X.W., W.Z. and Y.Q. added contents to Section Methods. F.W., Z.Y., N.D., Y.Q., S.H. and J.C. discussed the scope and content of the theoretical discussion. F.W. developed the optimization framework, and Z.Y. and Y.L. proposed the optimal control framework. N.D. verified the formula derivation. Y.Q. led the empirical study part. Y.Q., G.Y., Y.C., Y.S., W.C., J.Y., C.-M.C. and N.D. drafted Section Results. Y.Q., G.Y., W.C., J.Y. and S.H. conducted the experiments for overall performance and combination in Section Results. Y.S. and C.-M.C. conducted and wrote experiments for transferability and power of scale in Section Results. S.H. and Y.Q. drafted the application part. Z.L., H.-T.Z, Y.L., J.T., J.L. and M.S. advised the project, suggested the theoretical and empirical study and participated in the discussion. N.D. and Y.Q. participated in all the sections and proofread the whole paper.

Competing interests

The authors declare no competing interests.

Additional information

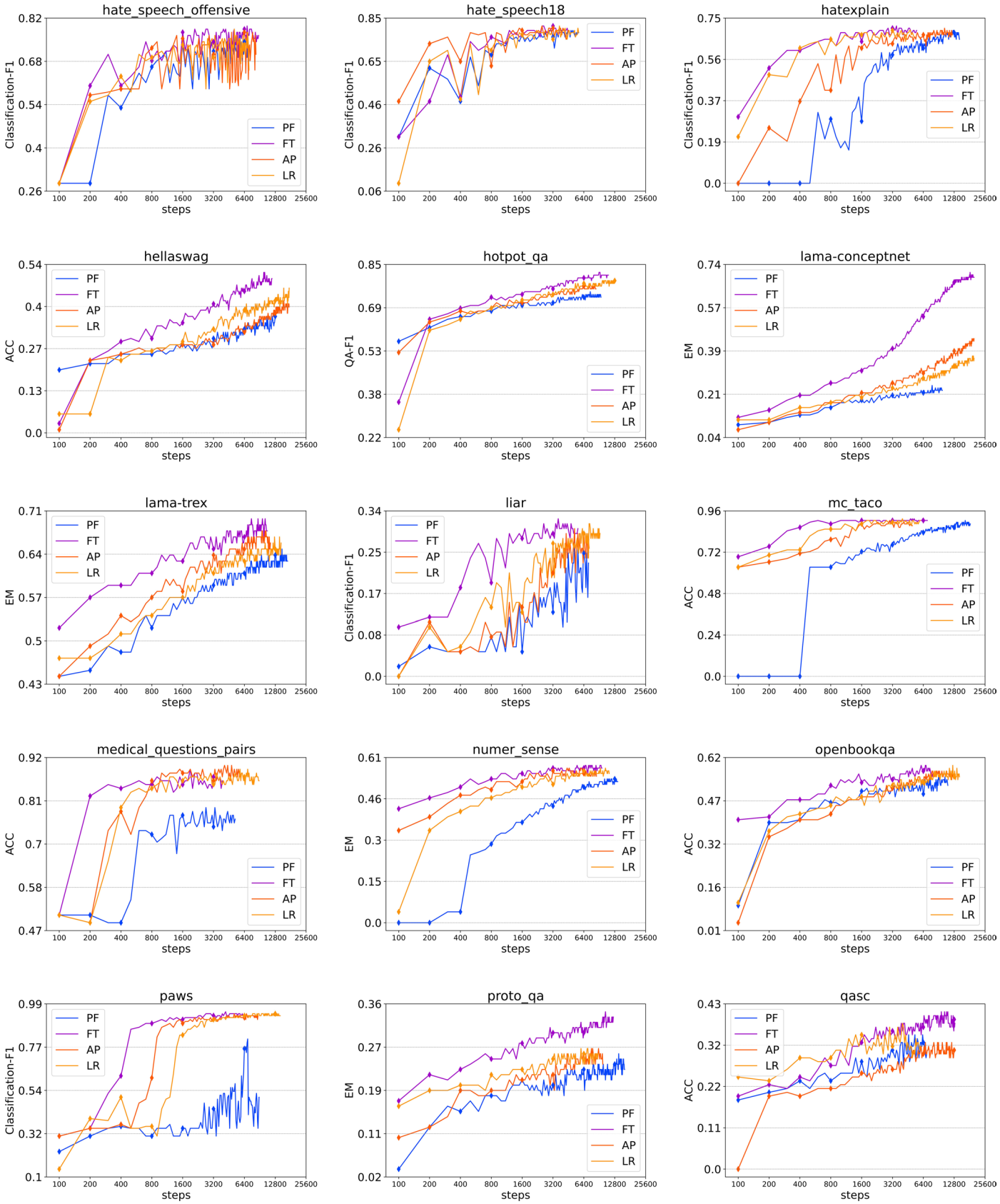
Extended data is available for this paper at https://doi.org/10.1038/s42256-023-00626-4.

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s42256-023-00626-4.

Correspondence and requests for materials should be addressed to Zhiyuan Liu, Hai-Tao Zheng or Maosong Sun.

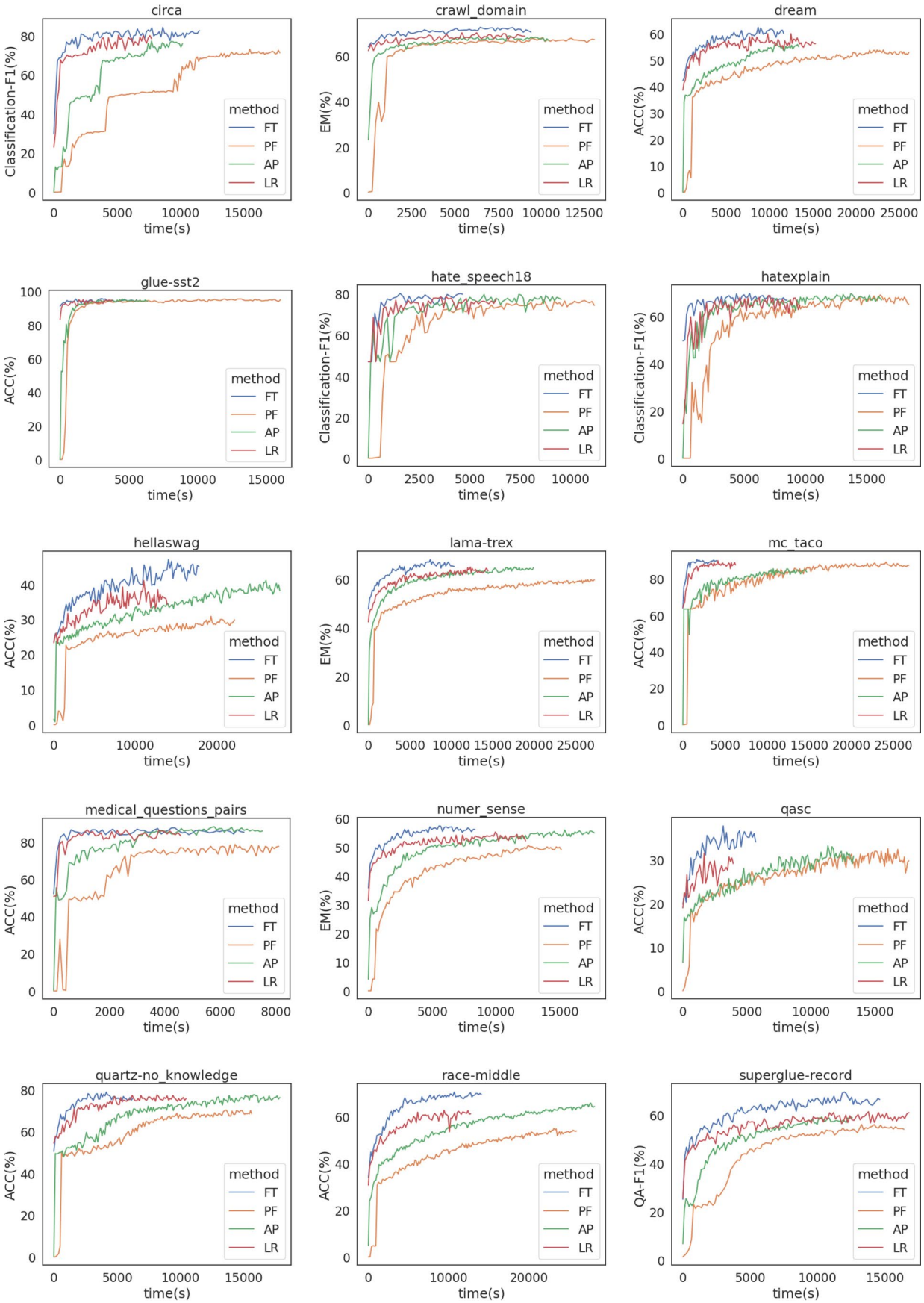
Peer review information *Nature Machine Intelligence* thanks Dieuwke Hupkes and the other, anonymous, reviewer(s) for their contribution to the peer review of this work.

https://doi.org/10.1038/s42256-023-00626-4



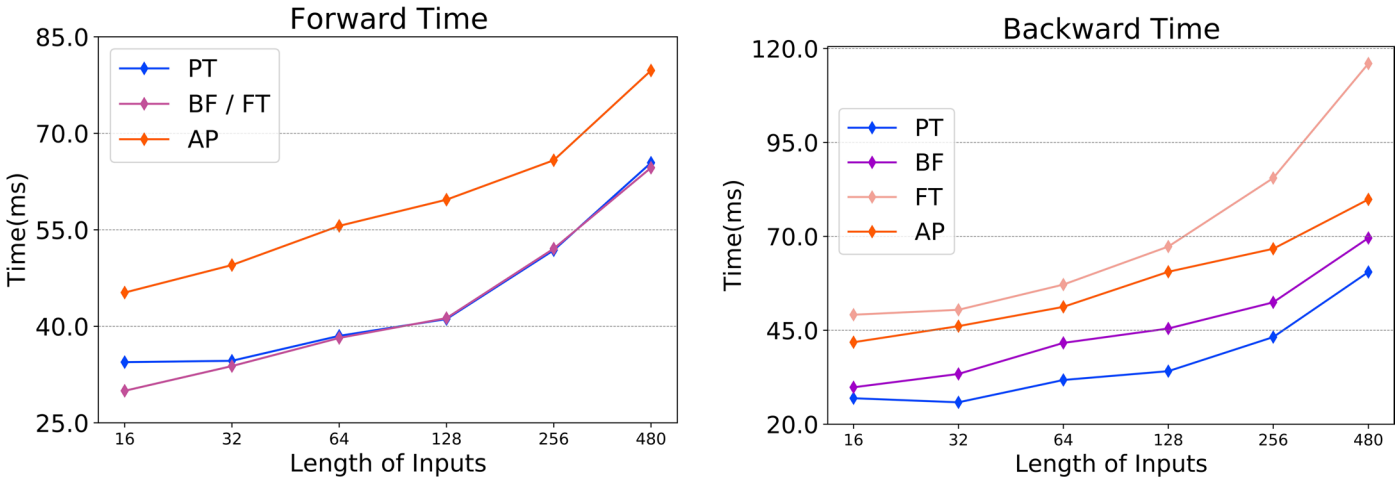
Extended Data Fig. 1 | The performance of $T5_{BASE}$ with different delta-tuning methods at different training steps. The performance of $T5_{BASE}$ with different delta-tuning methods (LR, AP, PF) and fine-tuning (FT) at different training steps.

https://doi.org/10.1038/s42256-023-00626-4



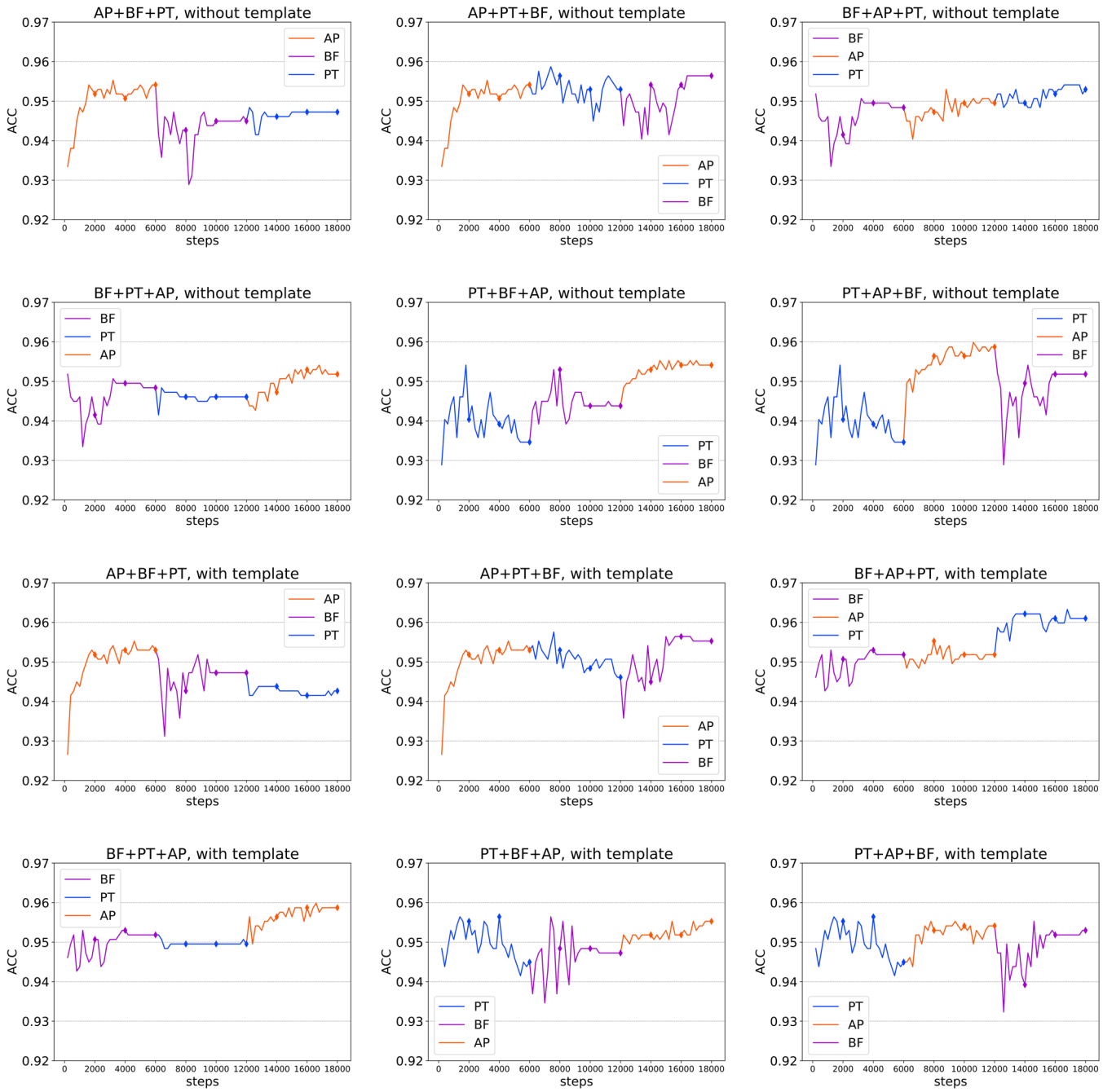
Extended Data Fig. 2 | The performance of $T5_{BASE}$ with different delta-tuning methods at different training time. The performance of $T5_{BASE}$.with different delta-tuning methods (LR, AP, PF) and fine-tuning (FT) at different training time (seconds).

https://doi.org/10.1038/s42256-023-00626-4



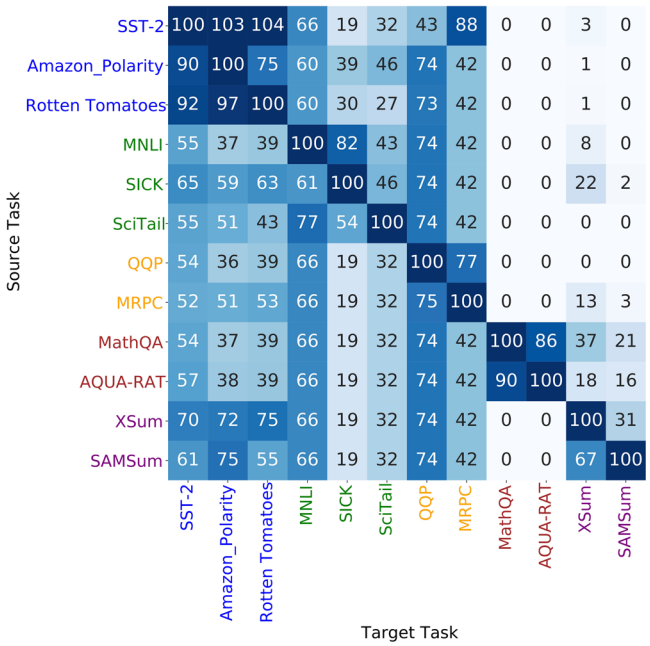
Extended Data Fig. 3 | Time consumption for fine-tuning (FT) and different delta-tuning methods. Time consumption for fine-tuning (FT) and different delta-tuning methods, including BitFit (BF), adapter (AP) and prompt-tuning (PT). We report the results with different input length.

https://doi.org/10.1038/s42256-023-00626-4

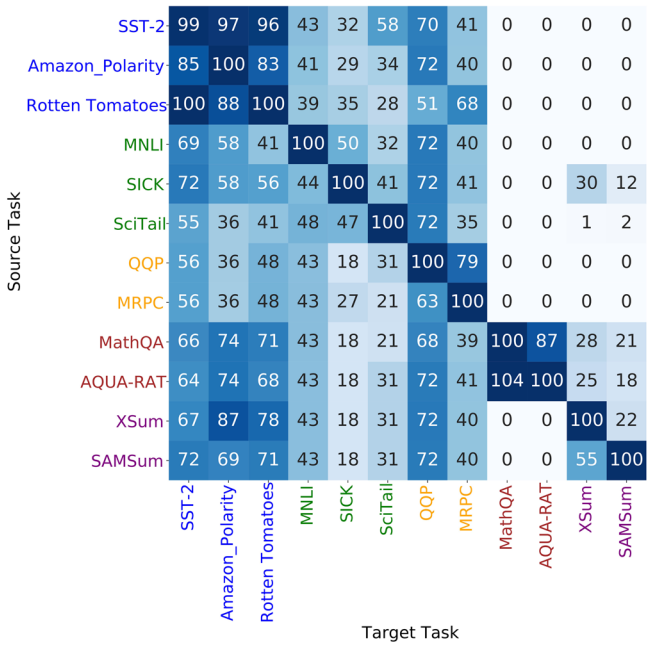


Extended Data Fig. 4 | The performance of RoBERTa$_{\text{LARGE}}$ when sequentially applying different delta-tuning methods. The performance of RoBERTa$_{\text{LARGE}}$ when different delta-tuning methods (adapter (AP), BitFit (BF) and prompt-tuning (PT)) are applied sequentially. The experiments are conducted on SST-2.

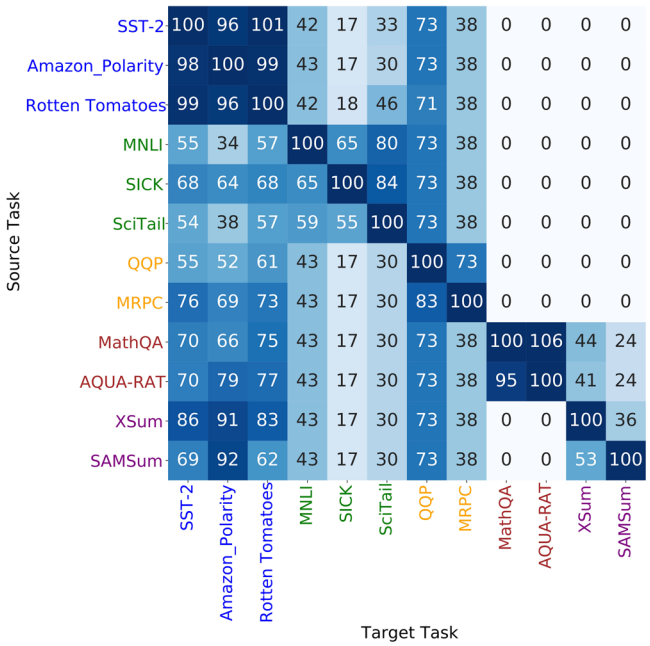
https://doi.org/10.1038/s42256-023-00626-4



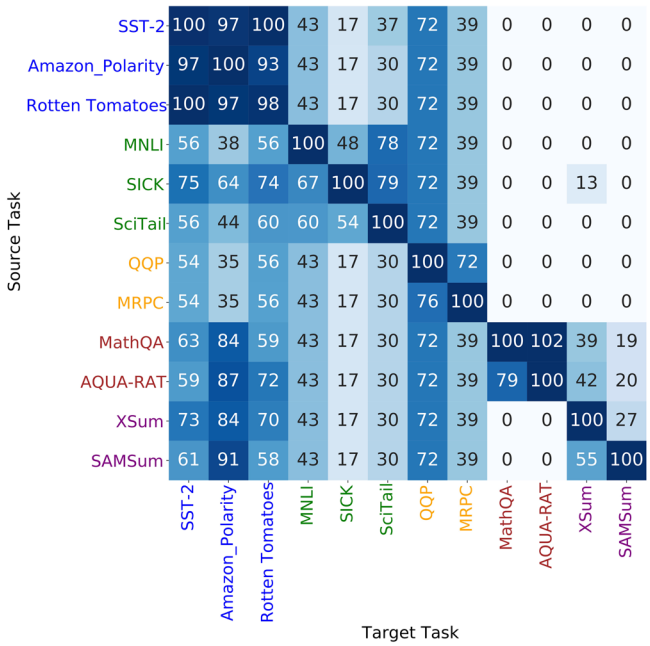
(a) Prompt Tuning



(b) Prefix-tuning



(c) Adapter



(d) LoRA

Extended Data Fig. 5 | Zero-shot transferring performance of four delta-tuning methods using $T5_{BASE}$. Zero-shot transferring performance of four delta-tuning methods using $T5_{BASE}$. We report relative performance (zero-shot transferring performance / original performance) (%) on the target tasks (columns) when delta parameters are transferred from the source tasks (rows). Colours of the task names indicate the task types. Blue: sentiment analysis. Green: natural language inference. Orange: paraphrase identification. Brown: question answering. Purple: summarization.

https://doi.org/10.1038/s42256-023-00626-4

Extended Data Table 1 | Statistics of the usage of different sizes of pre-trained models

Venue	No PLMs	Small PLMs	Large PLMs	Per. of Large PLMs
ACL 2022	26	167	7	3.5%
ACL 2021	41	151	8	4.0%
EMNLP 2021	46	150	4	2.0%
NAACL 2021	37	158	5	2.5%
ACL 2020	107	92	1	0.5%
EMNLP 2020	62	137	1	0.5%

The usage of models of different sizes in research published in NLP conferences, the statistic is based on 1000 randomly selected papers. Large PLMs are defined as PLMs with over 1 billion parameters.

https://doi.org/10.1038/s42256-023-00626-4

Extended Data Table 2 | Performance for $T5_{BASE}$ on GLUE datasets

Prompt	✗	✗	✗	✗	✓	✓	✓	✓
BitFit	✗	✗	✓	✓	✗	✗	✓	✓
Adapter	✗	✓	✗	✓	✗	✓	✗	✓
Tunable parameters	0%	0.89%	0.09%	0.98%	0.003%	0.893%	0.093%	0.983%
$T5_{BASE}$, *full-data, without manual templates*								
CoLA(Matt.)	-	**59.2**$_{0.2}$	58.7$_{1.7}$	58.4$_{0.8}$	34.0$_{18.6}$	51.2$_{4.6}$	36.9$_{21.7}$	57.8$_{0.3}$
SST-2(acc)	-	94.6$_{0.1}$	94.4$_{0.1}$	95.0$_{0.2}$	94.0$_{0.3}$	94.1$_{1.4}$	95.0$_{0.1}$	**95.1**$_{0.2}$
MRPC(F1)	-	89.1$_{0.6}$	90.1$_{0.4}$	**90.8**$_{0.3}$	84.8$_{1.2}$	89.2$_{0.7}$	88.5$_{0.7}$	88.8$_{0.6}$
STS-B(Pear.)	-	86.7$_{0.3}$	86.6$_{0.1}$	**86.9**$_{0.3}$	83.1$_{1.8}$	86.1$_{0.4}$	85.8$_{1.4}$	85.8$_{0.4}$
QQP(F1)	-	86.7$_{0.2}$	**88.3**$_{0.1}$	87.7$_{0.3}$	83.4$_{1.0}$	86.9$_{0.4}$	88.0$_{0.4}$	88.1$_{0.2}$
MNLI(acc)	-	84.5$_{0.4}$	87.1$_{0.3}$	87.1$_{0.4}$	81.6$_{0.2}$	86.0$_{0.2}$	**87.3**$_{0.2}$	87.1$_{0.4}$
QNLI(acc)	-	89.8$_{0.1}$	91.6$_{0.1}$	91.3$_{0.1}$	87.8$_{0.3}$	89.1$_{0.3}$	**91.8**$_{0.3}$	91.7$_{0.2}$
RTE(acc)	-	75.3$_{1.0}$	77.5$_{1.3}$	**80.0**$_{0.8}$	64.3$_{0.9}$	71.5$_{4.7}$	72.9$_{6.6}$	71.5$_{2.0}$
Average	-	83.2$_{0.3}$	84.3$_{0.5}$	**84.7**$_{0.4}$	76.6$_{3.1}$	81.8$_{1.6}$	80.8$_{3.9}$	83.2$_{0.5}$
$T5_{BASE}$, *full-data, with manual templates*								
CoLA(Matt.)	-	57.4$_{1.8}$	**59.5**$_{1.2}$	59.1$_{0.4}$	36.3$_{18.2}$	39.1$_{22.8}$	58.1$_{0.6}$	48.2$_{7.4}$
SST-2(acc)	-	95.0$_{0.1}$	94.8$_{0.3}$	95.0$_{0.2}$	93.9$_{0.1}$	95.2$_{0.1}$	95.0$_{0.2}$	**95.3**$_{0.3}$
MRPC(F1)	-	90.9$_{0.3}$	90.7$_{0.6}$	**91.4**$_{0.4}$	81.3$_{3.5}$	87.8$_{0.5}$	89.4$_{0.3}$	89.2$_{0.4}$
STS-B(Pear.)	-	87.1$_{0.2}$	87.4$_{0.4}$	**87.7**$_{0.2}$	83.4$_{1.0}$	86.6$_{0.1}$	84.7$_{3.4}$	86.8$_{0.4}$
QQP(F1)	-	87.4$_{0.1}$	**88.3**$_{0.1}$	88.2$_{0.1}$	83.7$_{1.1}$	87.4$_{0.1}$	88.3$_{0.1}$	88.3$_{0.2}$
MNLI(acc)	-	86.1$_{0.2}$	87.1$_{0.2}$	86.7$_{0.5}$	83.1$_{0.5}$	86.3$_{0.4}$	**87.2**$_{0.4}$	86.9$_{0.3}$
QNLI(acc)	-	91.9$_{0.3}$	92.8$_{0.4}$	92.6$_{0.2}$	89.3$_{0.7}$	92.2$_{0.1}$	**92.9**$_{0.1}$	92.7$_{0.3}$
RTE(acc)	-	81.9$_{0.7}$	**84.0**$_{0.8}$	83.2$_{1.5}$	66.8$_{2.2}$	80.0$_{0.2}$	81.8$_{1.3}$	80.1$_{0.6}$
Average	-	84.7$_{0.5}$	**85.6**$_{0.5}$	85.5$_{0.4}$	77.2$_{3.4}$	81.8$_{3.0}$	84.7$_{0.8}$	83.4$_{1.2}$

Performance of $T5_{BASE}$ on GLUE datasets. We report the average result of multiple random seeds on the validation set. ✓ denotes the component is included in the combination and ✗ denotes it is excluded in the combination.

https://doi.org/10.1038/s42256-023-00626-4

Extended Data Table 3 | Generalization gap for RoBERTa$_{\text{LARGE}}$ on GLUE datasets

Prompt	✗	✗	✗	✓	✓	✓	✓	
BitFit	✗	✓	✓	✗	✗	✓	✓	**FT**
Adapter	✓	✗	✓	✗	✓	✗	✓	
Tunable parameters	1.75%	0.09%	1.84%	0.003%	1.76%	0.09%	1.85%	100%
RoBERTa$_{\text{BASE}}$, *full-data, without manual templates*								
CoLA	$25.4_{1.5}$	$13.0_{2.8}$	$28.4_{2.4}$	$12.1_{3.8}$	$\mathbf{29.5}_{6.8}$	$16.2_{7.6}$	$18.0_{1.8}$	$28.2_{2.4}$
SST-2	$3.0_{1.3}$	$1.7_{0.5}$	$0.9_{0.3}$	$1.1_{0.5}$	$\mathbf{3.6}_{0.5}$	$1.9_{0.6}$	$3.5_{1.1}$	$3.3_{0.9}$
MRPC	$7.1_{0.3}$	$5.7_{2.2}$	$7.0_{0.4}$	$1.0_{1.1}$	$\mathbf{8.0}_{0.5}$	$4.5_{0.5}$	$7.1_{0.2}$	$6.3_{0.7}$
STS-B	$5.1_{0.0}$	$4.9_{0.6}$	$7.0_{0.8}$	$6.7_{1.6}$	$6.5_{0.3}$	$5.6_{0.6}$	$6.5_{0.4}$	$\mathbf{7.5}_{0.2}$
QQP	$0.6_{0.1}$	$0.7_{0.1}$	$0.8_{0.0}$	$0.1_{0.0}$	$0.8_{0.0}$	$0.7_{0.1}$	$0.8_{0.1}$	$\mathbf{1.9}_{0.2}$
MNLI	$\mathbf{0.6}_{0.1}$	$0.5_{0.1}$	$0.6_{0.2}$	$0.6_{0.4}$	$0.5_{0.1}$	$0.5_{0.2}$	$0.5_{0.1}$	$0.6_{0.0}$
QNLI	$0.9_{0.1}$	$0.7_{0.1}$	$0.5_{0.2}$	$1.6_{0.1}$	$0.5_{0.2}$	$0.8_{0.3}$	$0.5_{0.2}$	$\mathbf{1.6}_{0.0}$
RTE	$13.1_{0.6}$	$13.2_{0.7}$	$14.9_{0.3}$	$9.8_{1.6}$	$\mathbf{15.9}_{0.8}$	$12.6_{2.3}$	$15.1_{1.3}$	$12.9_{1.3}$
Average	$7.0_{0.5}$	$5.1_{0.9}$	$7.5_{0.6}$	$4.1_{1.1}$	$\mathbf{8.2}_{1.2}$	$5.3_{1.5}$	$6.5_{0.7}$	$7.8_{0.7}$
RoBERTa$_{\text{BASE}}$, *full-data, with manual templates*								
CoLA	$20.9_{5.0}$	$25.4_{4.4}$	$24.3_{7.5}$	$11.4_{1.2}$	$29.1_{3.2}$	$29.6_{6.4}$	$24.6_{10.3}$	$\mathbf{30.4}_{2.3}$
SST-2	$3.3_{0.1}$	$1.4_{0.6}$	$1.3_{0.7}$	$1.0_{0.3}$	$2.6_{0.7}$	$2.5_{0.8}$	$3.8_{0.4}$	$\mathbf{4.0}_{0.1}$
MRPC	$6.2_{2.5}$	$6.5_{0.6}$	$6.4_{0.3}$	$3.8_{2.5}$	$\mathbf{8.2}_{0.2}$	$7.2_{0.3}$	$6.7_{1.4}$	$7.2_{0.5}$
STS-B	$5.8_{1.4}$	$4.9_{0.4}$	$6.7_{1.2}$	$\mathbf{10.2}_{0.6}$	$6.9_{0.5}$	$5.5_{0.7}$	$6.1_{1.5}$	$7.5_{0.2}$
QQP	$0.7_{0.1}$	$0.6_{0.1}$	$0.8_{0.0}$	$0.2_{0.1}$	$0.8_{0.2}$	$0.7_{0.1}$	$0.8_{0.0}$	$\mathbf{2.0}_{0.1}$
MNLI	$\mathbf{0.8}_{0.1}$	$0.3_{0.1}$	$0.4_{0.1}$	$0.7_{0.2}$	$0.6_{0.1}$	$0.7_{0.1}$	$0.6_{0.1}$	$0.8_{0.2}$
QNLI	$0.8_{0.1}$	$0.4_{0.2}$	$0.1_{0.0}$	$1.4_{0.1}$	$0.1_{0.0}$	$0.5_{0.2}$	$0.0_{0.0}$	$\mathbf{2.0}_{0.1}$
RTE	$13.1_{0.2}$	$11.7_{0.8}$	$13.9_{0.7}$	$10.1_{5.2}$	$15.0_{0.4}$	$13.3_{1.3}$	$\mathbf{15.1}_{0.9}$	$12.7_{1.1}$
Average	$6.4_{1.2}$	$6.4_{0.9}$	$6.7_{1.3}$	$4.8_{1.3}$	$7.9_{0.7}$	$7.5_{1.2}$	$7.2_{1.8}$	$\mathbf{8.3}_{0.6}$

The experiments of generalization gap for RoBERTa$_{\text{LARGE}}$ on GLUE datasets. We report the average result (train performance - dev performance) of multiple random seeds. ✓ denotes the component is included in the combination and ✗ denotes it is excluded in the combination.

 https://doi.org/10.1038/s42256-023-00626-4

Extended Data Table 4 | Comparison between different delta-tuning methods

Name	Method	#Params
SEQUENTIAL ADAPTER	$\text{LayerNorm}(\mathbf{X}+\mathbf{H}(\mathbf{X})) \rightarrow \text{LayerNorm}(\mathbf{X}+\underline{\text{ADT}}(\mathbf{H}(\mathbf{X})))$	$L \times 2 \times (2d_h d_m)$
COMPACTER	$\text{LayerNorm}(\mathbf{X}+\mathbf{F}(\mathbf{X})) \rightarrow \text{LayerNorm}(\mathbf{X}+\underline{\text{ADT}}(\mathbf{F}(\mathbf{X})))$	$L \times 2 \times (2(d_h+d_m))$
ADAPTERDROP	$\underline{\text{ADT}}(\mathbf{X}) = \mathbf{X} + \sigma(\underline{\mathbf{X}\mathbf{W}_{d_h \times d_m}})\underline{\mathbf{W}_{d_m \times d_h}}, \ \sigma = \text{activation}$	$(L-n) \times 2 \times (2d_h d_m)$
PARALLEL ADAPTER	$\text{LayerNorm}(\mathbf{X}+\mathbf{H}(\mathbf{X})) \rightarrow \text{LayerNorm}(\mathbf{X}+\underline{\text{ADT}}(\mathbf{X})+\mathbf{H}(\mathbf{X}))$ $\text{LayerNorm}(\mathbf{X}+\mathbf{F}(\mathbf{X})) \rightarrow \text{LayerNorm}(\mathbf{X}+\underline{\text{ADT}}(\mathbf{X})+\mathbf{F}(\mathbf{X}))$ $\underline{\text{ADT}}(\mathbf{X}) = \sigma(\underline{\mathbf{X}\mathbf{W}_{d_h \times d_m}})\underline{\mathbf{W}_{d_m \times d_h}}, \ \sigma = \text{activation}$	$L \times 2 \times (2d_h d_m)$
ADAPTERBIAS	$\text{LayerNorm}(\mathbf{X}+\mathbf{F}(\mathbf{X})) \rightarrow \text{LayerNorm}(\underline{\text{ADT}}(\mathbf{X})+\mathbf{F}(\mathbf{X}))$ $\underline{\text{ADT}}(X) = \underline{\mathbf{X}\mathbf{W}_{d_h \times 1}}\ \underline{\mathbf{W}_{1 \times d_h}}$	$L \times 2 \times d_h$
PREFIX-TUNING	$\mathbf{H}_i = \text{ATT}(\mathbf{X}\mathbf{W}_q^{(i)}, [\underline{\text{MLP}_k^{(i)}(\mathbf{P}'_k)} : \mathbf{X}\mathbf{W}_k^{(i)}], [\underline{\text{MLP}_v^{(i)}(\mathbf{P}'_v)} : \mathbf{X}\mathbf{W}_v^{(i)}])$ $\underline{\text{MLP}^{(i)}}(\mathbf{X}) = \sigma(\underline{\mathbf{X}\mathbf{W}_{d_m \times d_m}})\underline{\mathbf{W}^{(i)}_{d_m \times d_h}}$ $\underline{P'} = \underline{\mathbf{W}_{n \times d_m}}$	$n \times d_m + d_m^2$ $+ L \times 2 \times d_h d_m$
LORA	$\mathbf{H}_i = \text{ATT}(\mathbf{X}\mathbf{W}_q^{(i)}, \underline{\text{ADT}_k}(\mathbf{X}) + \mathbf{X}\mathbf{W}_k^{(i)}, \underline{\text{ADT}_v}(\mathbf{X}) + \mathbf{X}\mathbf{W}_v^{(i)})$ $\underline{\text{ADT}}(\mathbf{X}) = \underline{\mathbf{X}\mathbf{W}_{d_h \times d_m}}\ \underline{\mathbf{W}_{d_m \times d_h}}$	$L \times 2 \times (2d_h d_m)$
BITFIT	$f(\mathbf{X}) \rightarrow f(\mathbf{X}) + \underline{\mathbf{B}}, \ \text{for all function } f$	$L \times (7 \times d_h + d_m)$

Comparison between different delta-tuning methods. we use underline to denote tunable parameters and modules. [:] is the concatenation operation; d_h means the hidden dimension of the Transformer model; d_m is the intermediate dimension between down-projection and up-projection, where d_m is far smaller than d_h. Compacter utilize hypercomplex matrix multiplication and low-rank decomposition to reduce the amount of parameters; AdapterDrop randomly dropout adapters in the first n layers and also bring down backpropagation time; Prefix-Tuning adds prefixes of n past key-values.